This guidebook applies to TI-Nspire™ software version 3.2. To obtain the latest version of the documentation, go to education.ti.com/guides.
Important Information

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

License

Please see the complete license installed in C:\Program Files\TI Education\<TI-Nspire™ Product Name>\license.

© 2006 - 2012 Texas Instruments Incorporated
Contents

Expression Templates
Fraction template ........................................ 1
Exponent template ........................................ 1
Square root template .................................... 1
Nth root template ......................................... 1
e^ exponent template .................................... 2
Log template .............................................. 2
Piecewise template (2-piece) ....................... 2
Piecewise template (N-piece) ....................... 2
System of 2 equations template ................... 3
System of N equations template ................... 3
Absolute value template ..................... 3
dd*nm’s.ss” template ................................. 3
Matrix template (2 x 2) ............................... 3
Matrix template (1 x 2) ............................... 4
Matrix template (2 x 1) ............................... 4
Matrix template (m x n) ....................... 4
Sum template (%) ........................................ 4
Product template (%) ................................... 4
First derivative template ............................ 5
Second derivative template ........................ 5
Definite integral template .......................... 5

Alphabetical Listing

A
abs() ............................................. 6
amortTbl() ...................................... 6
and ............................................ 6
angle() ......................................... 7
ANOVA ......................................... 7
ANOVA2way ...................................... 8
Ans .......................................... 9
approx() ....................................... 10
>approxFraction() ......................... 10
approxRational() .......................... 10
arccos() ....................................... 10
arccosh() ...................................... 10
arccot() ........................................ 10
arccoth() ....................................... 11
arccsc() ....................................... 11
arccsch() ...................................... 11
arccsc() ....................................... 11
arccsch() ...................................... 11
arcsin() ....................................... 11
arcsinh() ...................................... 11
arctan() ....................................... 11
arctanh() ...................................... 11
augment() ...................................... 11
avgRC() ........................................ 12

B
bal() ........................................... 12
>Base2 ......................................... 12
>Base10 ........................................ 13
>Base16 ....................................... 14
binomCdf() .................................... 14
binomPdf() .................................... 14

C
celiling() ......................................... 14
centralDiff() .................................... 15
char() ........................................... 15
χ^2way ........................................... 15
χ^2cdf() .......................................... 16
χ^2GOF ........................................... 16
χ^2Pdf() .......................................... 16
ClearAZ ......................................... 16
ClaErr ........................................... 17
colAugment() .................................... 17
colDim() ......................................... 17
colNorm() ........................................ 17
completeSquare() ................................ 18
cos() .......................................... 19
cos() .......................................... 19
cos() .......................................... 20
cosh() .......................................... 21
cosh() .......................................... 21
cot() .......................................... 21
cot() .......................................... 22
coth() .......................................... 22
coth() .......................................... 22

count() ......................................... 22
count() ......................................... 22
count() ......................................... 22
cPolyRoots() .................................... 23
crossP() ........................................ 23
csc() .......................................... 24
csc() .......................................... 24
csc() .......................................... 24
csc() .......................................... 24
CubicReg ......................................... 25
cumulativeSum() .................................. 25
Cycle ............................................ 26
>Cylind ......................................... 26

D
dbd() ........................................... 26
>DD ............................................... 27
>Decimal ........................................ 27
Define .......................................... 27
Define LibPriv .................................. 28
Define LibPub ................................... 28
deltaList() ...................................... 29
DelVar .......................................... 29
delVoid() ...................................... 29
det() .......................................... 29
det() .......................................... 30
diag() .......................................... 30
dim() .......................................... 30
Disp ........................................... 30
>DMS ........................................... 31
dotP() .......................................... 31

E
e^() ........................................... 31
eff() .......................................... 32
| P | propFrac() ................................................... | 78 |
|   | prRx() .......................................................... | 73 |
|   | prRy() .......................................................... | 74 |
|   | PassErr .......................................................... | 74 |
|   | piecewise() ..................................................... | 74 |
|   | poissCdf() ..................................................... | 74 |
|   | poissPdf() ..................................................... | 74 |
|   | Polar ............................................................ | 75 |
|   | polyEval() ........................................................ | 75 |
|   | polyRoots() ...................................................... | 75 |
|   | PowerReg ........................................................ | 76 |
|   | Prgm .............................................................. | 77 |
|   | prodSeq() ......................................................... | 77 |
|   | Product (PI) ....................................................... | 77 |
|   | product() ........................................................ | 77 |
|   | propFrac() ....................................................... | 78 |
| Q | QR ................................................................. | 78 |
|   | QuadReg ........................................................... | 79 |
|   | QuartReg .......................................................... | 79 |
| R | Rp0() ............................................................... | 80 |
|   | RpPr() ............................................................. | 80 |
|   | ›Rad ................................................................. | 81 |
|   | rand() .............................................................. | 81 |
|   | randBin() .......................................................... | 81 |
|   | randint() .......................................................... | 81 |
|   | randMat() .......................................................... | 81 |
|   | randNorm() ........................................................ | 81 |
|   | randPoly() ........................................................ | 82 |
|   | randSamp() ........................................................ | 82 |
|   | RandSeed .......................................................... | 82 |
|   | real() .............................................................. | 82 |
|   | ›Rect ................................................................. | 82 |
|   | ref() ............................................................... | 83 |
|   | remain() ........................................................... | 84 |
|   | Request ............................................................ | 84 |
|   | RequestStr ........................................................ | 85 |
|   | Return .............................................................. | 85 |
|   | right() ............................................................. | 85 |
|   | rk23() ............................................................... | 86 |
|   | root() .............................................................. | 86 |
|   | rotate() ............................................................ | 86 |
|   | round() ............................................................ | 87 |
|   | rowAdd() ........................................................... | 87 |
|   | rowDim() ........................................................... | 88 |
|   | rowNorm() .......................................................... | 88 |
|   | rowSwap() .......................................................... | 88 |
|   | rref() ............................................................. | 88 |
| S | sec() ............................................................... | 89 |
|   | sec'( ) ............................................................... | 89 |
|   | sech() ............................................................. | 89 |
|   | sech'( ) ............................................................. | 89 |
|   | seq() ............................................................... | 90 |
|   | seqGen() ........................................................... | 90 |
|   | seqn() ............................................................. | 91 |
|   | setMode() .......................................................... | 91 |
|   | shift() ............................................................. | 92 |
|   | sign() .............................................................. | 93 |
|   | simult() ........................................................... | 93 |
|   | sin() ............................................................... | 94 |
|   | sin'( ) ............................................................... | 94 |
|   | sinh() ............................................................. | 95 |
|   | sinh'( ) ............................................................. | 95 |
|   | SinReg ............................................................ | 96 |
|   | SortA ............................................................... | 96 |
|   | SortD ............................................................... | 97 |
|   | ›Sphere .............................................................. | 97 |
|   | sqrt() .............................................................. | 97 |
|   | stat.results ........................................................ | 98 |
|   | stat.values ........................................................ | 99 |
|   | stDevPop() ........................................................ | 99 |
|   | stDevSamp() ....................................................... | 99 |
|   | Stop ............................................................... | 100 |
|   | Store .............................................................. | 100 |
|   | string() ........................................................... | 100 |
|   | subMat() ........................................................... | 100 |
|   | Sum (Sigma) ....................................................... | 100 |
|   | sum() .............................................................. | 100 |
|   | sumIf() ............................................................. | 101 |
|   | sumSeq() ........................................................... | 101 |
|   | system() ........................................................... | 101 |
| T | T (transpose) ..................................................... | 101 |
|   | tan() ............................................................... | 102 |
|   | tan'( ) ............................................................... | 102 |
|   | tanh() ............................................................. | 103 |
|   | tanh'( ) ............................................................. | 103 |
|   | tCdf() .............................................................. | 104 |
|   | Text ............................................................... | 104 |
|   | Then ............................................................... | 104 |
|   | tInterval .......................................................... | 104 |
|   | tInterval_2Samp ................................................ | 105 |
|   | tPdf() ............................................................. | 105 |
|   | trace() ............................................................. | 105 |
|   | Try ............................................................... | 106 |
|   | tTest .............................................................. | 106 |
|   | tTest_2Samp ...................................................... | 107 |
|   | tvmFV() ............................................................ | 107 |
|   | tvmIF() ............................................................ | 108 |
|   | tvmN() ............................................................. | 108 |
|   | tvmPmt() .......................................................... | 108 |
|   | tvmPV() ........................................................... | 108 |
|   | TwoVar ............................................................ | 109 |
| U | unitV() ............................................................ | 110 |
|   | unLock ............................................................. | 110 |
| V | varPop() ........................................................... | 110 |
|   | varSamp() .......................................................... | 111 |
| W | warnCodes() ....................................................... | 111 |
|   | when() ............................................................ | 111 |
|   | While ............................................................. | 112 |
| X | xor ............................................................... | 112 |
This guide lists the templates, functions, commands, and operators available for evaluating math expressions.

**Expression Templates**

Expression templates give you an easy way to enter math expressions in standard mathematical notation. When you insert a template, it appears on the entry line with small blocks at positions where you can enter elements. A cursor shows which element you can enter.

Use the arrow keys or press \tab to move the cursor to each element’s position, and type a value or expression for the element. Press \enter or \ctrl\enter to evaluate the expression.

### Fraction template

![Fraction template](image)

**Example:**

\[
\begin{array}{c}
\frac{12}{8} \\
\frac{2}{4}
\end{array}
\]

*Note:* See also \( / \) (divide), page 118.

### Exponent template

![Exponent template](image)

**Example:**

\[
2^3 = 8
\]

*Note:* Type the first value, press \(^\), and then type the exponent.

To return the cursor to the baseline, press right arrow (\( \triangleright \)).

*Note:* See also \(^\) (power), page 119.

### Square root template

![Square root template](image)

**Example:**

\[
\begin{array}{c}
\sqrt{4} = 2 \\
\sqrt{\{9,4\}} = \{3,2\} \\
\sqrt{4} = 2 \\
\sqrt{\{9,16,4\}} = \{3,4,2\}
\end{array}
\]

*Note:* See also \( \sqrt{} \) (square root), page 125.

### Nth root template

![Nth root template](image)

**Example:**

\[
\begin{array}{c}
\sqrt[3]{8} = 2 \\
\sqrt[3]{\{8,27,15\}} = \{2,3,4\} \\
\sqrt[3]{8,27,15} = \{2,3,4\}
\end{array}
\]

*Note:* See also \( \text{root()} \), page 86.
\[ e \text{ exponent template} \]

\[ e^x \text{ keys} \]

Natural exponential \( e \) raised to a power

**Note:** See also \( e^x() \), page 31.

\[ \text{Log template} \]

\[ \log \left( \frac{1}{x} \right) \text{ key} \]

Calculates log to a specified base. For a default of base 10, omit the base.

**Note:** See also \( \log() \), page 58.

\[ \text{Piecewise template (2-piece)} \]

\[ \text{Catalog} > \]

Lets you create expressions and conditions for a two-piece piecewise function. To add a piece, click in the template and repeat the template.

**Note:** See also \( \text{piecewise}() \), page 74.

\[ \text{Piecewise template (N-piece)} \]

\[ \text{Catalog} > \]

Lets you create expressions and conditions for an \( N \)-piece piecewise function. Prompts for \( N \).

**Example:**

See the example for Piecewise template (2-piece).

**Note:** See also \( \text{piecewise}() \), page 74.
System of 2 equations template

\[
\begin{align*}
x + y &= 0 \\
x - y &= 5
\end{align*}
\]

Creates a system of two linear equations. To add a row to an existing system, click in the template and repeat the template.

**Note:** See also system(), page 101.

Example:

\[
\begin{align*}
solve(x + y &= 0, x, y) \\
solve(x - y &= 5, x, y)
\end{align*}
\]

\[x = \frac{5}{2} \text{ and } y = \frac{-5}{2}\]

\[
\begin{align*}
x &= \frac{-3}{2} \\
y &= \frac{1}{4}
\end{align*}
\]

System of N equations template

Lets you create a system of \(N\) linear equations. Prompts for \(N\).

**Note:** See also system(), page 101.

Example:

See the example for System of equations template (2-equation).

Absolute value template

**Note:** See also abs(), page 6.

Example:

\[
\begin{align*}
2, -3, 4, -4\end{align*}
\]

\[
\begin{align*}
\{2, 3, 4, 6.4\}
\end{align*}
\]

dd°mm'ss.ss'' template

Lets you enter angles in \(\text{dd}^\circ\text{mm}'\text{ss.ss}''\) format, where \(\text{dd}\) is the number of decimal degrees, \(\text{mm}\) is the number of minutes, and \(\text{ss.ss}\) is the number of seconds.

Example:

\[
\begin{align*}
30^\circ15'10'' \\
0.528011
\end{align*}
\]

Matrix template (2 x 2)

Creates a 2 x 2 matrix.

Example:

\[
\begin{bmatrix}
1 & 2 & 5 \\
3 & 4 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 & 10 \\
15 & 20
\end{bmatrix}
\]
Matrix template (1 x 2)  

Example: 

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

Matrix template (2 x 1)  

Example: 

\[
\begin{bmatrix}
5 \\
8
\end{bmatrix}
\begin{bmatrix}
0.01 \\
0.08
\end{bmatrix}
\]

Matrix template (m x n)  

The template appears after you are prompted to specify the number of rows and columns.  

Example: 

\[
\begin{bmatrix}
4 & 2 & 6 \\
1 & 2 & 3 \\
5 & 7 & 9
\end{bmatrix}
\]

Note: If you create a matrix with a large number of rows and columns, it may take a few moments to appear.

Sum template (\(\Sigma\))  

Example: 

\[
\sum_{n=3}^{7} (n) = 25
\]

Note: See also \(\Sigma()\) (sumSeq), page 126.

Product template (\(\Pi\))  

Example: 

\[
\prod_{n=1}^{5} \left(\frac{1}{n}\right) = \frac{1}{120}
\]

Note: See also \(\Pi()\) (prodSeq), page 125.
First derivative template

\[
\frac{d}{dx}(\square)
\]

The first derivative template can be used to calculate first derivative at a point numerically, using auto differentiation methods.

Note: See also d() (derivative), page 124.

Second derivative template

\[
\frac{d^2}{dx^2}(\square)
\]

The second derivative template can be used to calculate second derivative at a point numerically, using auto differentiation methods.

Note: See also d() (derivative), page 124.

Definite integral template

\[
\int_{\square}^{\square} d\square
\]

The definite integral template can be used to calculate the definite integral numerically, using the same method as nInt().

Note: See also nInt(), page 68.
Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, starting on page 117. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

abs()

Returns the absolute value of the argument.

Note: See also Absolute value template, page 3.

If the argument is a complex number, returns the number’s modulus.

amortTbl()

Amortization function that returns a matrix as an amortization table for a set of TVM arguments.

NPmt is the number of payments to be included in the table. The table starts with the first payment.

N, I, PV, FV, Pmt, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 108.

- If you omit Pmt, it defaults to Pmt=tvmPmt(N, I, PV, FV, PpY, CpY, PmtAt).
- If you omit FV, it defaults to FV=0.
- The defaults for PpY, CpY, and PmtAt are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.

The columns in the result matrix are in this order: Payment number, amount paid to interest, amount paid to principal, and balance.

The balance displayed in row n is the balance after payment n.

You can use the output matrix as input for the other amortization functions \( \Sigma\text{Int()} \) and \( \Sigma\text{Prn()} \), page 126, and \( \text{bal()} \), page 12.

and

Returns true or false or a simplified form of the original entry.
**Integer1 and Integer2 → integer**

Compares two real integers bit-by-bit using an **and** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

In **Hex base mode**:

| 0b7AC36 and 0b3D5F | 0h2C16 |

**Important**: Zero, not the letter O.

In **Bin base mode**:

| 0b100101 and 0b100 | 0b100 |

In **Dec base mode**:

| 37 and 0b100 | 4 |

**Note**: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

---

**angle()**

angle(Value1) → value

Returns the angle of the argument, interpreting the argument as a complex number.

**In Degree angle mode**:

| angle(0+2·i) | 90 |

**In Gradian angle mode**:

| angle(0+3·i) | 100 |

**In Radian angle mode**:

| angle(1+i) | 0.785398 |
| angle(1+2·i,3+0·i,0−4·i) | \{1.10715,0.,−1.5708\} |
| angle(1+2·i,3+0·i,0−4·i) | \{\frac{\pi}{2},\tan^{-1}(\frac{1}{2}),0,\frac{\pi}{2}\} |

---

**ANOVA**

ANOVA List1,List2[,List3,...,List20][,

Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the **stat.results** variable. (See page 98.)

**Flag=0 for Data, Flag=1 for Stats**

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>Value of the F statistic</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom of the groups</td>
</tr>
<tr>
<td>stat.SS</td>
<td>Sum of squares of the groups</td>
</tr>
<tr>
<td>stat.MS</td>
<td>Mean squares for the groups</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
</tbody>
</table>
Outputs: Block Design

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.SSError</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Mean square for the errors</td>
</tr>
<tr>
<td>stat.sp</td>
<td>Pooled standard deviation</td>
</tr>
<tr>
<td>stat.xbarlist</td>
<td>Mean of the input of the lists</td>
</tr>
<tr>
<td>stat.CLowerList</td>
<td>95% confidence intervals for the mean of each input list</td>
</tr>
<tr>
<td>stat.CUpperList</td>
<td>95% confidence intervals for the mean of each input list</td>
</tr>
</tbody>
</table>

**ANOVA2way**

**ANOVA2way** List1,List2[,List3,...,List10] LevRow

Computes a two-way analysis of variance for comparing the means of two to 10 populations. A summary of results is stored in the stat.results variable. (See page 98.)

LevRow=0 for Block

LevRow=2,3,...,Len-1, for Two Factor, where Len=length(List1)=length(List2) = = length(List10) and Len / LevRow ∈ {2,3, }

Outputs: Block Design

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>( F ) statistic of the column factor</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td>stat.SS</td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td>stat.MS</td>
<td>Mean squares for column factor</td>
</tr>
<tr>
<td>stat.FBlock</td>
<td>( F ) statistic for factor</td>
</tr>
<tr>
<td>stat.PValBlock</td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.dfBlock</td>
<td>Degrees of freedom for factor</td>
</tr>
<tr>
<td>stat.SSBlock</td>
<td>Sum of squares for factor</td>
</tr>
<tr>
<td>stat.MSBlock</td>
<td>Mean squares for factor</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSError</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Mean squares for the errors</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>

COLUMN FACTOR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.Fcol</td>
<td>( F ) statistic of the column factor</td>
</tr>
</tbody>
</table>
### ROW FACTOR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.PValCol</td>
<td>Probability value of the column factor</td>
</tr>
<tr>
<td>stat.dfCol</td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td>stat.SSCol</td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td>stat.MSCol</td>
<td>Mean squares for column factor</td>
</tr>
</tbody>
</table>

### INTERACTION Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.FRow</td>
<td>F statistic of the row factor</td>
</tr>
<tr>
<td>stat.PValRow</td>
<td>Probability value of the row factor</td>
</tr>
<tr>
<td>stat.dfRow</td>
<td>Degrees of freedom of the row factor</td>
</tr>
<tr>
<td>stat.SSRow</td>
<td>Sum of squares of the row factor</td>
</tr>
<tr>
<td>stat.MSRow</td>
<td>Mean squares for row factor</td>
</tr>
</tbody>
</table>

### ERROR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSErr</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSErr</td>
<td>Mean squares for the errors</td>
</tr>
<tr>
<td>s</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>

### Ans

<table>
<thead>
<tr>
<th>Ans</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ans → value</td>
<td>Returns the result of the most recently evaluated expression.</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>56 ÷ 4</td>
<td>60</td>
</tr>
<tr>
<td>60 ÷ 4</td>
<td>64</td>
</tr>
</tbody>
</table>
### `approx()`

**approx(\text{Value}) \Rightarrow \text{number}**

Returns the evaluation of the argument as an expression containing decimal values, when possible, regardless of the current Auto or Approximate mode.

This is equivalent to entering the argument and pressing `ctrl enter`.

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{approx}\left(\frac{1}{3}\right))</td>
<td>0.333333</td>
</tr>
<tr>
<td>(\text{approx}\left(\frac{1}{3},9\right))</td>
<td>({0.333333,0.111111})</td>
</tr>
<tr>
<td>(\text{approx}\left(\sin(\pi),\cos(\pi)\right))</td>
<td>({0.,1.})</td>
</tr>
<tr>
<td>(\text{approx}\left(\sqrt{2},\sqrt{3}\right))</td>
<td>(\begin{bmatrix} 1.41421 &amp; 1.73205 \end{bmatrix})</td>
</tr>
<tr>
<td>(\text{approx}\left(\frac{1}{3},9\right))</td>
<td>(\begin{bmatrix} 0.333333 &amp; 0.111111 \end{bmatrix})</td>
</tr>
<tr>
<td>(\text{approx}\left(\sin(\pi),\cos(\pi)\right))</td>
<td>({0.,1.})</td>
</tr>
<tr>
<td>(\text{approx}\left(\sqrt{2},\sqrt{3}\right))</td>
<td>(\begin{bmatrix} 1.41421 &amp; 1.73205 \end{bmatrix})</td>
</tr>
</tbody>
</table>

### `approxfrac()`

**approxfrac(\text{Value} \{\text{Tol}\}) \Rightarrow \text{value}**

**approxfrac(\text{List} \{\text{Tol}\}) \Rightarrow \text{list}**

**approxfrac(\text{Matrix} \{\text{Tol}\}) \Rightarrow \text{matrix}**

Returns the input as a fraction, using a tolerance of \(\text{Tol}\). If \(\text{Tol}\) is omitted, a tolerance of 5.E-14 is used.

**Note:** You can insert this function from the computer keyboard by typing `\texttt{@>\text{approxfrac}(\ldots)}`.

### `approxrational()`

**approxrational(\text{Value}, \text{Tol}) \Rightarrow \text{value}**

**approxrational(\text{List}, \text{Tol}) \Rightarrow \text{list}**

**approxrational(\text{Matrix}, \text{Tol}) \Rightarrow \text{matrix}**

Returns the argument as a fraction using a tolerance of \(\text{Tol}\). If \(\text{Tol}\) is omitted, a tolerance of 5.E-14 is used.

### `arccos()`

See \(\cos^{-1}(\text{\ldots})\), page 20.

### `arccosh()`

See \(\cosh^{-1}(\text{\ldots})\), page 21.

### `arccot()`

See \(\cot^{-1}(\text{\ldots})\), page 22.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arccoth()</td>
<td>See coth⁻¹(0), page 22.</td>
</tr>
<tr>
<td>arccsc()</td>
<td>See csc⁻¹(0), page 24.</td>
</tr>
<tr>
<td>arccsch()</td>
<td>See csch⁻¹(0), page 24.</td>
</tr>
<tr>
<td>arcsec()</td>
<td>See sec⁻¹(0), page 89.</td>
</tr>
<tr>
<td>arcsech()</td>
<td>See sech⁻¹(0), page 89.</td>
</tr>
<tr>
<td>arcsin()</td>
<td>See sin⁻¹(0), page 94.</td>
</tr>
<tr>
<td>arcsinh()</td>
<td>See sinh⁻¹(0), page 95.</td>
</tr>
<tr>
<td>arctan()</td>
<td>See tan⁻¹(0), page 102.</td>
</tr>
<tr>
<td>arctanh()</td>
<td>See tanh⁻¹(0), page 103.</td>
</tr>
</tbody>
</table>

**augment()**

**augment(List1, List2) ⇒ list**

Returns a new list that is List2 appended to the end of List1.

**augment(Matrix1, Matrix2) ⇒ matrix**

Returns a new matrix that is Matrix2 appended to Matrix1. When the "," character is used, the matrices must have equal row dimensions, and Matrix2 is appended to Matrix1 as new columns. Does not alter Matrix1 or Matrix2.

```
| 1 2 | → m1  |  1 2 |
| 3 4 |      |  3 4 |
| 5   | → m2  |  5   |
| 6   |      |  6   |
```

```
augment(m1,m2)  |  1 2 5 |
               |  3 4 6 |
```
**avgRC()**

`avgRC(Expr1, Var [= Value], Step) ⇒ expression`

`avgRC(Expr1, Var [= Value], List1) ⇒ list`

`avgRC(List1, Var [= Value], Step) ⇒ list`

`avgRC(Matrix1, Var [= Value], Step) ⇒ matrix`

Returns the forward-difference quotient (average rate of change). `Expr1` can be a user-defined function name (see `Func`).

When `Value` is specified, it overrides any prior variable assignment or any current `|` substitution for the variable.

`Step` is the step value. If `Step` is omitted, it defaults to 0.001.

Note that the similar function `centralDiff()` uses the central-difference quotient.

---

**bal()**

`bal(NPmt, N, I, PV, FV, PpY, CpY, PmtAt, [roundValue]) ⇒ value`

`bal(NPmt, amortTable) ⇒ value`

Amortization function that calculates schedule balance after a specified payment.

`N, I, PV, FV, Pmt, PpY, CpY,`, and `PmtAt` are described in the table of TVM arguments, page 108.

`NPmt` specifies the payment number after which you want the data calculated.

`N, I, PV, FV, Pmt, PpY, CpY,`, and `PmtAt` are described in the table of TVM arguments, page 108.

- If you omit `Pmt`, it defaults to `Pmt=tvmPmt(N, I, PV, FV, PpY, CpY, PmtAt)`.
- If you omit `FV`, it defaults to `FV=0`.
- The defaults for `PpY, CpY,` and `PmtAt` are the same as for the TVM functions.

`roundValue` specifies the number of decimal places for rounding. Default=2.

`bal(NPmt, amortTable)` calculates the balance after payment number `NPmt`, based on amortization table `amortTable`. The `amortTable` argument must be a matrix in the form described under `amortTbl()`, page 6.

**Note:** See also `ΣInt()` and `ΣPrn()`, page 126.

---

**Base2**

`Integer1 Base2 ⇒ integer`

**Note:** You can insert this operator from the computer keyboard by typing `@>Base2`.

Converts `Integer1` to a binary number. Binary or hexadecimal numbers always have a `0b` or `0h` prefix, respectively.
Without a prefix, Integer1 is treated as decimal (base 10). The result is displayed in binary, regardless of the Base mode.

Negative numbers are displayed in “two’s complement” form. For example,

\(-1\) is displayed as:
- 0FFFFFFFFFFFFFFF in Hex base mode
- 0b111...111 (64 1’s) in Binary base mode

\(-2^{63}\) is displayed as:
- 0h8000000000000000 in Hex base mode
- 0b100...000 (63 zeros) in Binary base mode

If you enter a decimal integer that is outside the range of a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. Consider the following examples of values outside the range:

\(2^{63}\) becomes \(-2^{63}\) and is displayed as:
- 0h8000000000000000 in Hex base mode
- 0b100...000 (63 zeros) in Binary base mode

\(2^{64}\) becomes 0 and is displayed as:
- 0h0 in Hex base mode
- 0b0 in Binary base mode

\(-2^{63} - 1\) becomes \(2^{63} - 1\) and is displayed as:
- 0h7FFFFFFFFFFFFFFF in Hex base mode
- 0b111...111 (64 1’s) in Binary base mode
Base16

$Integer1 \rightarrow Base16 \Rightarrow integer$

**Note:** You can insert this operator from the computer keyboard by typing $@>Base16$.

Converts $Integer1$ to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

- 0b $binaryNumber$
- 0h $hexadecimalNumber$

Zero, not the letter O, followed by b or h.

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, $Integer1$ is treated as decimal (base 10). The result is displayed in hexadecimal, regardless of the Base mode.

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see Base2, page 12.

binomCdf()

$binomCdf(n,p) \Rightarrow number$

$binomCdf(n,p,lowBound,upBound) \Rightarrow number$ if $lowBound$ and $upBound$ are numbers, list if $lowBound$ and $upBound$ are lists

$binomCdf(n,p,upBound)$ for $P(0 \leq X \leq upBound) \Rightarrow number$ if $upBound$ is a number, list if $upBound$ is a list

Computes a cumulative probability for the discrete binomial distribution with $n$ number of trials and probability $p$ of success on each trial.

For $P(X \leq upBound)$, set $lowBound=0$

binomPdf()

$binomPdf(n,p) \Rightarrow number$

$binomPdf(n,p,XVal) \Rightarrow number$ if $XVal$ is a number, list if $XVal$ is a list

Computes a probability for the discrete binomial distribution with $n$ number of trials and probability $p$ of success on each trial.

ceiling()

$ceiling(Value1) \Rightarrow value$

Returns the nearest integer that is $\geq$ the argument.

The argument can be a real or a complex number.

**Note:** See also floor().

$ceiling(List1) \Rightarrow list$

$ceiling(Matrix1) \Rightarrow matrix$

Returns a list or matrix of the ceiling of each element.

---

256$\rightarrow Base16 \Rightarrow 0h100$

0b111100001111$\rightarrow Base16 \Rightarrow 0hF0F$

ceiling{-.3.1,1,2.5} $\Rightarrow \{-3.1,3.\}$

ceiling{0, -3.2, i} $\Rightarrow \{0, -3.1, i\}$

ceiling{{1.3, 4}} $\Rightarrow \{2, 4\}$
centralDiff()  

centralDiff(Expr1, Var [= Value [,Step]]) \rightarrow expression  

centralDiff(Expr1, Var [,Step])|Var=Value \rightarrow expression  

centralDiff(Expr1, Var [= Value [,List]]) \rightarrow list  

centralDiff(List1, Var [= Value [,Step]]) \rightarrow list  

centralDiff(Matrix1, Var [= Value [,Step]]) \rightarrow matrix  

Returns the numerical derivative using the central difference quotient formula.  

When Value is specified, it overrides any prior variable assignment or any current "|" substitution for the variable.  

Step is the step value. If Step is omitted, it defaults to 0.001.  

When using List1 or Matrix1, the operation gets mapped across the values in the list or across the matrix elements.  

Note: See also avgRC().  

char()  

char(Integer) \rightarrow character  

Returns a character string containing the character numbered Integer from the handheld character set. The valid range for Integer is 0–65535.  

\[ 
\chi^2 \text{2way} 
\]  

\[ 
\chi^2 \text{2way} \, \text{obsMatrix} 
\]  

\[ 
\chi^2 \text{2way} \, \text{obsMatrix} 
\]  

chi22way obsMatrix  

Computes a \(\chi^2\) test for association on the two-way table of counts in the observed matrix obsMatrix. A summary of results is stored in the stat.results variable. (See page 98.)  

For information on the effect of empty elements in a matrix, see "Empty (Void) Elements" on page 132.  

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.(\chi^2)</td>
<td>Chi square stat: sum (observed - expected)^2/expected</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom for the chi square statistics</td>
</tr>
<tr>
<td>stat.ExpMat</td>
<td>Matrix of expected elemental count table, assuming null hypothesis</td>
</tr>
<tr>
<td>stat.CompMat</td>
<td>Matrix of elemental chi square statistic contributions</td>
</tr>
</tbody>
</table>
\( \chi^2 \text{Cdf()} \)

\( \chi^2 \text{Cdf}(lowBound, upBound, df) \Rightarrow \text{number if } lowBound \text{ and } upBound \text{ are numbers, list if } lowBound \text{ and } upBound \text{ are lists} \\
\text{chi2Cdf} \Rightarrow \text{number if } lowBound \text{ and } upBound \text{ are numbers, list if } lowBound \text{ and } upBound \text{ are lists} \\
Computes the \( \chi^2 \) distribution probability between \( lowBound \) and \( upBound \) for the specified degrees of freedom \( df \).

For \( P(X \leq upBound) \), set \( lowBound = 0 \).

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

\( \chi^2 \text{GOF} \)

\( \chi^2 \text{GOF obsList, expList, df} \\
\text{chi2GOF obsList, expList, df} \\
\) Performs a test to confirm that sample data is from a population that conforms to a specified distribution. \( obsList \) is a list of counts and must contain integers. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat} \chi^2 )</td>
<td>Chi square stat: ( \sum(\text{observed} - \text{expected})^2/\text{expected} )</td>
</tr>
<tr>
<td>( \text{stat.PVal} )</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>( \text{stat.df} )</td>
<td>Degrees of freedom for the chi square statistics</td>
</tr>
<tr>
<td>( \text{stat.Cmplist} )</td>
<td>Elemental chi square statistic contributions</td>
</tr>
</tbody>
</table>

\( \chi^2 \text{Pdf()} \)

\( \chi^2 \text{Pdf}(XVal, df) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list} \\
\text{chi2Pdf(} \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list} \\
Computes the probability density function (pdf) for the \( \chi^2 \) distribution at a specified \( XVal \) value for the specified degrees of freedom \( df \).

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

ClearAZ

Clears all single-character variables in the current problem space.

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See \texttt{unLock}, page 110.
**ClrErr**

Clears the error status and sets system variable `errCode` to zero.

The **Else** clause of the **Try...Else...EndTry** block should use `ClrErr` or `PassErr`. If the error is to be processed or ignored, use `ClrErr`. If what to do with the error is not known, use `PassErr` to send it to the next error handler. If there are no more pending **Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.

**Note:** See also **PassErr**, page 74, and **Try**, page 106.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `@` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

For an example of `ClrErr`, See Example 2 under the **Try** command, page 106.

---

**colAugment()**

Returns a new matrix that is `Matrix2` appended to `Matrix1`. The matrices must have equal column dimensions, and `Matrix2` is appended to `Matrix1` as new rows. Does not alter `Matrix1` or `Matrix2`.

<table>
<thead>
<tr>
<th><code>colAugment(Matrix1, Matrix2)</code></th>
<th><code>matrix</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Matrix1</code></td>
<td><code>1 2</code></td>
</tr>
<tr>
<td></td>
<td><code>3 4</code></td>
</tr>
<tr>
<td></td>
<td><code>5 6</code></td>
</tr>
<tr>
<td><code>Matrix2</code></td>
<td><code>1 2</code></td>
</tr>
<tr>
<td></td>
<td><code>3 4</code></td>
</tr>
<tr>
<td></td>
<td><code>5 6</code></td>
</tr>
</tbody>
</table>

---

**colDim()**

Returns the number of columns contained in `Matrix`.

**Note:** See also `rowDim()`.

```
colDim(Matrix1) → expression
```

```
colDim(0 1 2)
```

```
3
```

---

**colNorm()**

Returns the maximum of the sums of the absolute values of the elements in the columns in `Matrix`.

**Note:** Undefined matrix elements are not allowed. See also `rowNorm()`.

```
colNorm(Matrix)
```

```
[1  2  3] → mat
[4  5  6]
```

```
colNorm(mat)
```

```
9
```
**completeSquare()**

**completeSquare(ExprOrEqn, Var) ⇒ expression or equation**

**completeSquare(ExprOrEqn, Var^Power) ⇒ expression or equation**

**completeSquare(ExprOrEqn, Var1 Var2 [...]) ⇒ expression or equation**

**completeSquare(ExprOrEqn, {Var1 Var2 [...]})) ⇒ expression or equation**

Converts a quadratic polynomial expression of the form $a\cdot x^2+b\cdot x+c$ into the form $a\cdot(x-h)^2+k$

- or -

Converts a quadratic equation of the form $a\cdot x^2+b\cdot x+c=d$ into the form $a\cdot(x-h)^2=k$

The first argument must be a quadratic expression or equation in standard form with respect to the second argument.

The second argument must be a single univariate term or a single univariate term raised to a rational power, for example $x$, $y^2$, or $z^{2/3}$.

The third and fourth syntax attempt to complete the square with respect to variables $Var1, Var2 [...])$.

**conj()**

**conj(Value1) ⇒ value**

**conj(List1) ⇒ list**

**conj(Matrix1) ⇒ matrix**

Returns the complex conjugate of the argument.

**constructMat()**

**constructMat(Expr, Var1, Var2,numRows,numCols) ⇒ matrix**

Returns a matrix based on the arguments.

$Expr$ is an expression in variables $Var1$ and $Var2$. Elements in the resulting matrix are formed by evaluating $Expr$ for each incremented value of $Var1$ and $Var2$.

$Var1$ is automatically incremented from 1 through $numRows$. Within each row, $Var2$ is incremented from 1 through $numCols$.

**CopyVar**

**CopyVar Var1, Var2**

**CopyVar Var1, Var2.**

**CopyVar Var1, Var2 copies the value of variable Var1 to variable Var2, creating Var2 if necessary. Variable Var1 must have a value.**

If $Var1$ is the name of an existing user-defined function, copies the definition of that function to function $Var2$. Function $Var1$ must be defined.

$Var1$ must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.
**CopyVar**

**CopyVar** \( Var_1, \ Var_2 \). copies all members of the \( Var_1 \) variable group to the \( Var_2 \) group, creating \( Var_2 \) if necessary.

\( Var_1 \) must be the name of an existing variable group, such as the statistics \( \text{stats} \) results, or variables created using the \( \text{LibShortcut}() \) function. If \( Var_2 \) already exists, this command replaces all members that are common to both groups and adds the members that do not already exist. If one or more members of \( Var_2 \) are locked, all members of \( Var_2 \) are left unchanged.

**corrMat()**

**corrMat**(\( List_1, List_2, ... [List_20] \))

Computes the correlation matrix for the augmented matrix \( [List_1, List_2, ... , List_20] \).

**cos()**

\( \cos(Value) \) \( \Rightarrow \) value

\( \cos(List) \) \( \Rightarrow \) list

\( \cos(Value) \) returns the cosine of the argument as a value.

\( \cos(List) \) returns a list of the cosines of all elements in \( List \).

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, ′, ″ or \( \hat{r} \) to override the angle mode temporarily.

---

<table>
<thead>
<tr>
<th><strong>Catalog &gt;</strong></th>
<th><strong>Done</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{aa.a} := 45 )</td>
<td>45</td>
</tr>
<tr>
<td>( \text{aa.b} := 6.78 )</td>
<td>6.78</td>
</tr>
<tr>
<td>( \text{aa.c} := 8.9 )</td>
<td>8.9</td>
</tr>
</tbody>
</table>

\[ \text{getVarInfo()} \]

\[ \text{aa.a alphanumeric} \]

\[ \text{aa.b alphanumeric} \]

<table>
<thead>
<tr>
<th><strong>Catalog &gt;</strong></th>
<th><strong>Done</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{getVarInfo()} )</td>
<td></td>
</tr>
<tr>
<td>( \text{aa.a alphanumeric} )</td>
<td></td>
</tr>
<tr>
<td>( \text{aa.b alphanumeric} )</td>
<td></td>
</tr>
<tr>
<td>( \text{aa.c alphanumeric} )</td>
<td></td>
</tr>
</tbody>
</table>

A. \( \text{aa.a := } 45 \)

B. \( \text{aa.b := } 6.78 \)

C. \( \text{aa.c := } 8.9 \)

---

\( \cos\left(\frac{\pi}{4}\right) \) \( \Rightarrow \) 0.707107

\( \cos\left(\frac{\pi}{4}\hat{r}\right) \) \( \Rightarrow \) 0.707107

\( \cos\left(\{0,60,90\}\right) \) \( \Rightarrow \) \{1.0,0.5,0.0\}

\( \cos\left(\{0,50,100\}\right) \) \( \Rightarrow \) \{1.0,0.707107,0.0\}

**In Degree angle mode:**

\( \cos\left(\{0,60,90\}\right) \) \( \Rightarrow \) \{1.0,0.5,0.0\}

**In Radian angle mode:**

\( \cos\left(\frac{\pi}{4}\right) \) \( \Rightarrow \) 0.707107

\( \cos\left(\frac{\pi}{4}\hat{r}\right) \) \( \Rightarrow \) 0.707107
\textbf{\texttt{cos()}}

\texttt{cos(squareMatrix1) \rightarrow squareMatrix}

Returns the matrix cosine of \texttt{squareMatrix1}. This is not the same as calculating the cosine of each element.

When a scalar function \texttt{f(A)} operates on \texttt{squareMatrix1} (A), the result is calculated by the algorithm:

Compute the eigenvalues ($\lambda_i$) and eigenvectors ($V_i$) of A.

\texttt{squareMatrix1} must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

Form the matrices:

\[ B = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix} \quad \text{and} \quad X = [V_1, V_2, \ldots, V_n] \]

Then \( A = X B X^{-1} \text{ and } f(A) = X f(B) X^{-1}. \) For example, \( \cos(A) = X \cos(B) X^{-1} \) where:

\[ \cos(B) = \begin{bmatrix} \cos(\lambda_1) & 0 & \ldots & 0 \\ 0 & \cos(\lambda_2) & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & \cos(\lambda_n) \end{bmatrix} \]

All computations are performed using floating-point arithmetic.

\textbf{\texttt{cos^{	ext{-}1}()}}

\texttt{cos^{-1}(Value1) \rightarrow value}
\texttt{cos^{-1}(List1) \rightarrow list}

\texttt{cos^{-1}(Value1)} returns the angle whose cosine is \texttt{Value1}.

\texttt{cos^{-1}(List1)} returns a list of the inverse cosines of each element of \texttt{List1}.

\textbf{Note:} The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

\textbf{Note:} You can insert this function from the keyboard by typing \texttt{arc\_cos( \ldots )}.

\texttt{cos^{-1}(squareMatrix1) \rightarrow squareMatrix}

Returns the matrix inverse cosine of \texttt{squareMatrix1}. This is not the same as calculating the inverse cosine of each element. For information about the calculation method, refer to \texttt{cos()}.\n
\texttt{squareMatrix1} must be diagonalizable. The result always contains floating-point numbers.
cosh()\[\text{Catalog } \Rightarrow \text{cosh}\]

\[\text{cosh(Value1)} \Rightarrow \text{value}\]
\[\text{cosh(List1)} \Rightarrow \text{list}\]

\text{cosh(Value1)} \text{ returns the hyperbolic cosine of the argument.}

\text{cosh(List1)} \text{ returns a list of the hyperbolic cosines of each element of List1.}

\text{cosh(squareMatrix1)} \Rightarrow \text{squareMatrix}

\text{Returns the matrix hyperbolic cosine of squareMatrix1. This is not the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to cos().}

\text{squareMatrix1} \text{ must be diagonalizable. The result always contains floating-point numbers.}

\text{cosh}^{-1}()\[\text{Catalog } \Rightarrow \text{cosh}^{-1}\]

\[\text{cosh}^{-1}(\text{Value1}) \Rightarrow \text{value}\]
\[\text{cosh}^{-1}(\text{List1}) \Rightarrow \text{list}\]

\text{cosh}^{-1}(\text{Value1}) \text{ returns the inverse hyperbolic cosine of the argument.}

\text{cosh}^{-1}(\text{List1}) \text{ returns a list of the inverse hyperbolic cosines of each element of List1.}

\text{Note: You can insert this function from the keyboard by typing \text{arccosh(...).}}

\text{cosh}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}

\text{Returns the matrix inverse hyperbolic cosine of squareMatrix1. This is not the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to cos().}

\text{squareMatrix1} \text{ must be diagonalizable. The result always contains floating-point numbers.}

\text{cot()}\[\text{Catalog } \Rightarrow \text{key}\]

\[\text{cot(Value1)} \Rightarrow \text{value}\]
\[\text{cot(List1)} \Rightarrow \text{list}\]

\text{Returns the cotangent of Value1 or returns a list of the cotangents of all elements in List1.}

\text{Note: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, ‰, or \text{r} to override the angle mode temporarily.}
\( \cot^{-1}(Value1) \Rightarrow value \)
\( \cot^{-1}(List1) \Rightarrow list \)

Returns the angle whose cotangent is \( Value1 \) or returns a list containing the inverse cotangents of each element of \( List1 \).

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the keyboard by typing \( \text{arccot}(...) \).

In Degree angle mode:
\( \cot^{-1}(1) \)

In Gradian angle mode:
\( \cot^{-1}(1) \)

In Radian angle mode:
\( \cot^{-1}(1) \)

\( \text{coth}(\) \)

\( \text{coth}(Value1) \Rightarrow value \)
\( \text{coth}(List1) \Rightarrow list \)

Returns the hyperbolic cotangent of \( Value1 \) or returns a list of the hyperbolic cotangents of all elements of \( List1 \).

\( \text{coth}^{-1}(\) \)

\( \text{coth}^{-1}(Value1) \Rightarrow value \)
\( \text{coth}^{-1}(List1) \Rightarrow list \)

Returns the inverse hyperbolic cotangent of \( Value1 \) or returns a list containing the inverse hyperbolic cotangents of each element of \( List1 \).

**Note:** You can insert this function from the keyboard by typing \( \text{arccoth}(...) \).

\( \text{count}(\) \)

\( \text{count}(Value1 \text{ or } List1, \text{ or } List2 \text{...}) \Rightarrow value \)

Returns the accumulated count of all elements in the arguments that evaluate to numeric values.

Each argument can be an expression, value, list, or matrix. You can mix data types and use arguments of various dimensions.

For a list, matrix, or range of cells, each element is evaluated to determine if it should be included in the count.

Within the Lists & Spreadsheet application, you can use a range of cells in place of any argument.

Empty (void) elements are ignored. For more information on empty elements, see page 132.
**countif()**

**countif(List, Criteria) ⇒ value**

Counts the number of elements equal to 3.

Counts the number of elements equal to "def".

Counts 1 and 3.

Counts 3, 5, and 7.

Counts 1, 3, 7, and 9.

**cPolyRoots()**

**cPolyRoots(Poly, Var) ⇒ list**

The first syntax, `cPolyRoots(Poly, Var)`, returns a list of complex roots of polynomial `Poly` with respect to variable `Var`.

**cPolyRoots(ListOfCoeffs) ⇒ list**

The second syntax, `cPolyRoots(ListOfCoeffs)`, returns a list of complex roots for the coefficients in `ListOfCoeffs`.

**crossP()**

**crossP(List1, List2) ⇒ list**

Returns the cross product of `List1` and `List2` as a list.

**crossP(Vector1, Vector2) ⇒ vector**

Returns a row or column vector (depending on the arguments) that is the cross product of `Vector1` and `Vector2`.

Both `Vector1` and `Vector2` must be row vectors, or both must be column vectors. Both vectors must have equal dimension, and the dimension must be either 2 or 3.
### csc()

**Syntax:**
- `csc(Value1) → value`
- `csc(List1) → list`

Returns the cosecant of `Value1` or returns a list containing the cosecants of all elements in `List1`.

**Examples:**
- In Degree angle mode:
  - `csc(45)` → 1.4142
- In Gradian angle mode:
  - `csc(50)` → 1.4142
- In Radian angle mode:
  - `csc([1, π/2, π/3])` → `{1.1884, 1.1547}`

### csc⁻¹()

**Syntax:**
- `csc⁻¹(Value1) → value`
- `csc⁻¹(List1) → list`

Returns the angle whose cosecant is `Value1` or returns a list containing the inverse cosecants of each element of `List1`.

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Examples:**
- In Degree angle mode:
  - `csc⁻¹(1)` → 90
- In Gradian angle mode:
  - `csc⁻¹(1)` → 100
- In Radian angle mode:
  - `csc⁻¹([1, 4, 6])` → `{1.5708, 0.25268, 0.167448}`

### csch()

**Syntax:**
- `csch(Value1) → value`
- `csch(List1) → list`

Returns the hyperbolic cosecant of `Value1` or returns a list of the hyperbolic cosecants of all elements of `List1`.

**Examples:**
- `csch(3)` → 0.099822
- `csch([1, 2, 1, 4])` → `{0.850918, 0.248641, 0.036644}`

### csch⁻¹()

**Syntax:**
- `csch⁻¹(Value) → value`
- `csch⁻¹(List1) → list`

Returns the inverse hyperbolic cosecant of `Value1` or returns a list containing the inverse hyperbolic cosecants of each element of `List1`.

**Note:** You can insert this function from the keyboard by typing `arccsch(...)`.

**Examples:**
- `csch⁻¹(1)` → 0.881374
- `csch⁻¹([1, 2, 1, 3])` → `{0.881374, 0.459815, 0.32745}`
**CubicReg**

**CubicReg** \( X, Y \), [Freq], [Category, Include]

Computes the cubic polynomial regression \( y = a \cdot x\^3 + b \cdot x\^2 + c \cdot x + d \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

### Output variable

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression equation: ( a \cdot x^3 + b \cdot x^2 + c \cdot x + d )</td>
</tr>
<tr>
<td>( \text{stat.a, stat.b, stat.c, stat.d} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.R}^2 )</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>( \text{stat.Resid} )</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>( \text{stat.XReg} )</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq ), ( \text{Category List} ), and ( \text{Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.YReg} )</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq ), ( \text{Category List} ), and ( \text{Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.FreqReg} )</td>
<td>List of frequencies corresponding to ( \text{stat.XReg} ) and ( \text{stat.YReg} )</td>
</tr>
</tbody>
</table>

**cumulativeSum()**

\[ \text{cumulativeSum}() \Rightarrow \text{list} \]

Returns a list of the cumulative sums of the elements in \( \text{List1} \), starting at element 1.

\[ \text{cumulativeSum}() \Rightarrow \text{matrix} \]

Returns a matrix of the cumulative sums of the elements in \( \text{Matrix1} \). Each element is the cumulative sum of the column from top to bottom.

An empty (void) element in \( \text{List1} \) or \( \text{Matrix1} \) produces a void element in the resulting list or matrix. For more information on empty elements, see page 132.

\[
\text{cumulativeSum}\left(\{1,2,3,4\}\right) = \{1,3,6,10\}
\]

\[
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
\end{array} \Rightarrow m1
\]

\[
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{array}
\]

\[
\text{cumulativeSum}(m1) = \begin{array}{cc}
1 & 2 \\
4 & 6 \\
9 & 12 \\
\end{array}
\]
**Cycle**

Transfers control immediately to the next iteration of the current loop (For, While, or Loop).

*Note for entering the example:* In the Calculator application on the handheld, you can enter multi-line definitions by pressing `@` instead of `Enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

Function listing that sums the integers from 1 to 100 skipping 50.

```plaintext
Define g() = Func
  Local temp, i
  0 → temp
  For i, 1, 100, 1
    If i = 50
      Cycle
      temp + i → temp
    EndFor
    Return temp
  EndFunc
```

```
g()
g mund
```

$5000$

**Cylind**

Displays the row or column vector in cylindrical form $[r, \theta, z]$.

`Vector` must have exactly three elements. It can be either a row or a column.

```
Vector → Cylind
```

```
[2 2 3] → Cylind

[2.82843 \ \angle 0.785398 \ \ 3.]
```

**dbd()**

Returns the number of days between `date1` and `date2` using the actual-day-count method.

`date1` and `date2` can be numbers or lists of numbers within the range of the dates on the standard calendar. If both `date1` and `date2` are lists, they must be the same length.

`date1` and `date2` must be between the years 1950 through 2049.

You can enter the dates in either of two formats. The decimal placement differentiates between the date formats.

MM.DDYY (format used commonly in the United States)
DDMM.YY (format use commonly in Europe)

```
dbd()
dbd(date1, date2) ➞ value
```

```
dbd(12.3103, 1.0104)
dbd(1.0107, 6.0107)
dbd(3112.03, 101.04)
dbd(101.07, 106.07)
```

$1$

$151$

$1$

$151$
DD

Expr1 \text{DD} \Rightarrow \text{value}
List1 \text{DD} \Rightarrow \text{list}
Matrix1 \text{DD} \Rightarrow \text{matrix}

\textbf{Note:} You can insert this operator from the computer keyboard by typing \textit{@>DD}.

Returns the decimal equivalent of the argument expressed in degrees. The argument is a number, list, or matrix that is interpreted by the Angle mode setting in gradians, radians or degrees.

\begin{align*}
\text{In Degree angle mode:} & \\
\left(1.5^\circ\right) \text{DD} & = 1.5^\circ \\
\left(45^\circ22'14.3''\right) \text{DD} & = 45.3706^\circ \\
\left\{\{45^\circ22'14.3'',60^\circ00''\}\right\} \text{DD} & = \{45.3706^\circ,60^\circ\}
\end{align*}

\begin{align*}
\text{In Gradian angle mode:} & \\
1 \text{DD} & = 900 \circ \\
10 & = 1000 \circ
\end{align*}

\begin{align*}
\text{In Radian angle mode:} & \\
\left(1.5\right) \text{DD} & = 85.9437^\circ
\end{align*}

Decimal

Number1 \text{Decimal} \Rightarrow \text{value}
List1 \text{Decimal} \Rightarrow \text{value}
Matrix1 \text{Decimal} \Rightarrow \text{value}

\textbf{Note:} You can insert this operator from the computer keyboard by typing \textit{@>Decimal}.

Displays the argument in decimal form. This operator can be used only at the end of the entry line.

Define

\textbf{Define Var = Expression}

\textbf{Define Function(Param1, Param2, ...) = Expression}

Defines the variable \textit{Var} or the user-defined function \textit{Function}.

Parameters, such as \textit{Param1}, provide placeholders for passing arguments to the function. When calling a user-defined function, you must supply arguments (for example, values or variables) that correspond to the parameters. When called, the function evaluates \textit{Expression} using the supplied arguments.

\textit{Var} and \textit{Function} cannot be the name of a system variable or built-in function or command.

\textbf{Note:} This form of \textbf{Define} is equivalent to executing the expression: \textit{expression \rightarrow Function(Param1,Param2)}.

\begin{align*}
\text{Define } g(x,y) &= 2\cdot x - 3\cdot y & \text{Done} \\
g(1,2) & = -4 \\
1 \rightarrow a: 2 \rightarrow b: g(a,b) & = -4 \\
\text{Define } h(x) \text{ - when } x < 2, \cdot x - 3, \cdot x + 3 & \text{ Done} \\
h(-3) & = -9 \\
h(4) & = -5
\end{align*}
Define

Define Function(Param1, Param2, ...) = Func
Block
EndFunc

Define Program(Param1, Param2, ...) = Prgm
Block
EndPrgm

In this form, the user-defined function or program can execute a block of multiple statements.

Block can be either a single statement or a series of statements on separate lines. Block also can include expressions and instructions (such as If, Then, Else, and For).

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing $\rightarrow$ instead of $\cdot$ at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Note: See also Define LibPriv, page 28, and Define LibPub, page 28.

Define LibPriv

Define LibPriv Var = Expression
Define LibPriv Function(Param1, Param2, ...) = Expression
Define LibPriv Function(Param1, Param2, ...) = Func
Block
EndFunc
Define LibPriv Program(Param1, Param2, ...) = Prgm
Block
EndPrgm

Operates the same as Define, except defines a private library variable, function, or program. Private functions and programs do not appear in the Catalog.

Note: See also Define, page 27, and Define LibPub, page 28.

Define LibPub

Define LibPub Var = Expression
Define LibPub Function(Param1, Param2, ...) = Expression
Define LibPub Function(Param1, Param2, ...) = Func
Block
EndFunc
Define LibPub Program(Param1, Param2, ...) = Prgm
Block
EndPrgm

Operates the same as Define, except defines a public library variable, function, or program. Public functions and programs appear in the Catalog after the library has been saved and refreshed.

Note: See also Define, page 27, and Define LibPriv, page 28.
deltaList()

See \texttt{\textbf{\texttt{|List|}}}, page 55.

\textbf{DelVar} \ Var1, \ Var2, \ldots

\textbf{DelVar} \ Var.

Deletes the specified variable or variable group from memory.

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See \texttt{\textbf{\texttt{unLock}}}, page 110.

\textbf{DelVar} \ Var. deletes all members of the \texttt{Var} variable group (such as the statistics \texttt{stat.nn} results or variables created using the \texttt{LibShortcut()} function). The dot (.) in this form of the \textbf{DelVar} command limits it to deleting a variable group; the simple variable \texttt{Var} is not affected.

\textbf{delVoid()}

\textbf{delVoid} (List1) \Rightarrow \texttt{list}

Returns a list that has the contents of \texttt{List1} with all empty (void) elements removed.

For more information on empty elements, see page 132.

\textbf{det()}

\textbf{det} (squareMatrix[, Tolerance]) \Rightarrow \texttt{expression}

Returns the determinant of \texttt{squareMatrix}.

Optionally, any matrix element is treated as zero if its absolute value is less than \textit{Tolerance}. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, \textit{Tolerance} is ignored.

- If you use \texttt{ctrl enter} or set the \textit{Auto or Approximate} mode to Approximate, computations are done using floating-point arithmetic.
- If \textit{Tolerance} is omitted or not used, the default tolerance is calculated as:

\[
\text{1E-14} \cdot \max (\text{dim(squareMatrix)}) \cdot \text{rowNorm(squareMatrix)}
\]
**diag()**

**diag(List) ⇒ matrix**

**diag(rowMatrix) ⇒ matrix**

**diag(columnMatrix) ⇒ matrix**

Returns a matrix with the values in the argument list or matrix in its main diagonal.

**diag(squareMatrix) ⇒ rowMatrix**

Returns a row matrix containing the elements from the main diagonal of squareMatrix. 

*squareMatrix* must be square.

---

**dim()**

**dim(List) ⇒ integer**

Returns the dimension of List.

**dim(Matrix) ⇒ list**

Returns the dimensions of matrix as a two-element list {rows, columns}.

**dim(String) ⇒ integer**

Returns the number of characters contained in character string String.

---

**Disp**

**Disp [exprOrString1 [, exprOrString2] ...**

Displays the arguments in the Calculator history. The arguments are displayed in succession, with thin spaces as separators.

Useful mainly in programs and functions to ensure the display of intermediate calculations.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing [↵] at the end of each line. On the computer keyboard, hold down Alt and press Enter.
**DMS**

*Value DMS*  
*List DMS*  
*Matrix DMS*

**Note:** You can insert this operator from the computer keyboard by typing @>DMS.

Interprets the argument as an angle and displays the equivalent DMS (DDDDD°MM'SS.ss") number. See °, ′, ″ on page 128 for DMS (degree, minutes, seconds) format.

**Note:** DMS will convert from radians to degrees when used in radian mode. If the input is followed by a degree symbol °, no conversion will occur. You can use DMS only at the end of an entry line.

**dotP()**

**dotP(List1, List2) → expression**

Returns the "dot" product of two lists.

**dotP(Vector1, Vector2) → expression**

Returns the "dot" product of two vectors. Both must be row vectors, or both must be column vectors.

**E**

**e^()**

**e^(Value1) → value**

Returns e raised to the Value1 power.

**Note:** See also e exponent template, page 2.

**Note:** Pressing the key to display e^ is different from pressing the character E on the keyboard.

You can enter a complex number in re^θ polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

**e^(List1) → list**

Returns e raised to the power of each element in List1.

**e^(squareMatrix1) → squareMatrix**

Returns the matrix exponential of squareMatrix1. This is not the same as calculating e raised to the power of each element. For information about the calculation method, refer to cos().

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.
**eff()**

**eff(nominalRate, CpY) => value**

Financial function that converts the nominal interest rate `nominalRate` to an annual effective rate, given `CpY` as the number of compounding periods per year.

`nominalRate` must be a real number, and `CpY` must be a real number greater than 0.

**Note:** See also `nom()`, page 69.

**eigVc()**

**eigVc(squareMatrix) => matrix**

Returns a matrix containing the eigenvectors for a real or complex `squareMatrix`, where each column in the result corresponds to an eigenvalue. Note that an eigenvector is not unique; it may be scaled by any constant factor. The eigenvectors are normalized, meaning that if \( V = [x_1, x_2, \ldots, x_n] \), then:

\[
x_1^2 + x_2^2 + \ldots + x_n^2 = 1
\]

`squareMatrix` is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The `squareMatrix` is then reduced to upper Hessenberg form and the eigenvectors are computed via a Schur factorization.

In Rectangular Complex Format:

```
-1 2 5
3 -6 9
2 -5 7
```

\(-1\ 2\ 5\)

\(3\ -6\ 9\)

\(2\ -5\ 7\)

**eigVl()**

**eigVl(squareMatrix) => list**

Returns a list of the eigenvalues of a real or complex `squareMatrix`.

`squareMatrix` is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The `squareMatrix` is then reduced to upper Hessenberg form and the eigenvalues are computed from the upper Hessenberg matrix.

In Rectangular Complex Format Mode:

```
-1 2 5
3 -6 9
2 -5 7
```

\(-4.40941, 2.20471 + 0.763006i, 2.20471 - 0i\)

To see the entire result, press \(\uparrow\) and then use \(\downarrow\) and \(\rightarrow\) to move the cursor.
ElseIf

**If** BooleanExpr1 **Then**
  Block1

ElseIf BooleanExpr2 **Then**
  Block2

ElseIf BooleanExprN **Then**
  BlockN

EndIf

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `@` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

Define \( g(x) = \text{Func} \)

- **If** \( x \leq 5 \) **Then**
  - Return 5
- ElseIf \( x > 5 \) and \( x < 0 \) **Then**
  - Return \( -x \)
- ElseIf \( x \geq 0 \) and \( x \neq 10 \) **Then**
  - Return \( x \)
- ElseIf \( x = 10 \) **Then**
  - Return 3

EndIf
EndFunc

---

EndFor
See For, page 38.

EndFunc
See Func, page 40.

EndIf
See If, page 45.

EndLoop
See Loop, page 60.

EndPrgm
See Prgm, page 77.

EndTry
See Try, page 106.

EndWhile
See While, page 112.

---

*TI-Nspire™ Reference Guide* 33
**euler()**

**euler(Expr, Var, depVar, {Var0 VarMax}, depVar0, VarStep \[, eulerStep\]) \Rightarrow matrix**

**euler(SystemOfExpr, Var, ListOfDepVars, {Var0, VarMax},
ListOfDepVar0, VarStep \[, eulerStep\]) \Rightarrow matrix**

**euler(ListOfExpr, Var, ListOfDepVars, {Var0, VarMax},
ListOfDepVar0, VarStep \[, eulerStep\]) \Rightarrow matrix**

Uses the Euler method to solve the system
\[
\frac{d \text{depVar}}{d \text{Var}} = \text{Expr(Var, depVar)}
\]
with \(\text{depVar(Var0)}=\text{depVar0}\) on the interval \([\text{Var0}, \text{VarMax}]\). Returns a matrix whose first row defines the \(\text{Var}\) output values and whose second row defines the value of the first solution component at the corresponding \(\text{Var}\) values, and so on.

\(\text{Expr}\) is the right-hand side that defines the ordinary differential equation (ODE).

\(\text{SystemOfExpr}\) is the system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in \(\text{ListOfDepVars}\)).

\(\text{ListOfExpr}\) is a list of right-hand sides that define the system of ODEs (corresponds to the order of dependent variables in \(\text{ListOfDepVar}\)).

\(\text{Var}\) is the independent variable.

\(\text{ListOfDepVar}\) is a list of dependent variables.

\(\{\text{Var0, VarMax}\}\) is a two-element list that tells the function to integrate from \(\text{Var0}\) to \(\text{VarMax}\).

\(\text{ListOfDepVar0}\) is a list of initial values for dependent variables.

\(\text{VarStep}\) is a nonzero number such that \(\text{sign(VarMax-Var0)} = \text{sign(VarMax-Var0)}\) and solutions are returned at \(\text{Var0+k\VarStep}\) for all \(k=0,1,2,\ldots\) such that \(\text{Var0+k\VarStep}\) is in \([\text{Var0, VarMax}]\) (there may not be a solution value at \(\text{VarMax}\)).

\(\text{eulerStep}\) is a positive integer (defaults to 1) that defines the number of euler steps between output values. The actual step size used by the euler method is \(\text{VarStep/eulerStep}\).

**Differential equation:**
\[
y' = 0.001y(100-y) \text{ and } y(0)=10
\]

**System of equations:**
\[
y1' = y1+0.1y1y2 \text{ and } y1(0)=2
\]
\[
y2' = 3y2-y1y2 \text{ and } y2(0)=5
\]

```
euler\left(\begin{array}{c}
y1' + 0.1y1y2 \\
3y2-y1y2
\end{array}\right), \left\{\begin{array}{c}
y1, y2
\end{array}\right\}, \left\{\begin{array}{c}
0.5 \\\n2.5
\end{array}\right\}, 1
\right)
```

```
\begin{array}{cccccccc}
0. & 1. & 2. & 3. & 4. & 5. \\
\end{array}
```

```
\begin{array}{cccccccc}
\end{array}
```

**Exit**

**Exit**

Exits the current For, While, or Loop block.

**Exit** is not allowed outside the three looping structures (For, While, or Loop).

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing 
instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.
**exp()**

\[ \text{exp}(\text{Value1}) \Rightarrow \text{value} \]

Returns \( e \) raised to the \( \text{Value1} \) power.

**Note:** See also \( e \) exponent template, page 2.

You can enter a complex number in \( re^{i\theta} \) polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

\[ \text{exp}(\text{List1}) \Rightarrow \text{list} \]

Returns \( e \) raised to the power of each element in \( \text{List1} \).

\[ \text{exp}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

Returns the matrix exponential of \( \text{squareMatrix1} \). This is not the same as calculating \( e \) raised to the power of each element. For information about the calculation method, refer to \( \text{cos()} \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating point numbers.

**expr()**

\[ \text{expr}(\text{String}) \Rightarrow \text{expression} \]

Returns the character string contained in \( \text{String} \) as an expression and immediately executes it.

**ExpReg**

\[ \text{ExpReg } X, Y \left[ , \text{Freq} \right] \left[ , \text{Category} \right] \left[ , \text{Include} \right] \]

Computes the exponential regression \( y = a \cdot (b)^x \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression equation: ( a \cdot (b)^x )</td>
</tr>
<tr>
<td>( \text{stat.a} ), ( \text{stat.b} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.r}^2 )</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>( \text{stat.r} )</td>
<td>Correlation coefficient for transformed data (x, ln(y))</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals associated with the exponential model</td>
</tr>
<tr>
<td>stat.ResidTrans</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

### F

**factor()**

`factor(rationalNumber)` returns the rational number factored into primes. For composite numbers, the computing time grows exponentially with the number of digits in the second-largest factor. For example, factoring a 30-digit integer could take more than a day, and factoring a 100-digit number could take more than a century.

To stop a calculation manually,
- **Windows®**: Hold down the F12 key and press Enter repeatedly.
- **Macintosh®**: Hold down the F5 key and press Enter repeatedly.
- **Handheld**: Hold down the (a on) key and press (enter) repeatedly.

If you merely want to determine if a number is prime, use `isPrime()` instead. It is much faster, particularly if `rationalNumber` is not prime and if the second-largest factor has more than five digits.

**Fcdf()**

`Fcdf(lowBound,upBound,dfNumer,dfDenom) ⇒ number` if `lowBound` and `upBound` are numbers, list if `lowBound` and `upBound` are lists

Computes the F distribution probability between `lowBound` and `upBound` for the specified `dfNumer` (degrees of freedom) and `dfDenom`.

For $P(X \leq upBound)$, set `lowBound = 0`.

**Fill**

Replaces each element in variable `matrixVar` with `Value`. `matrixVar` must already exist.

```
1 2 → a matrix
3 4
```

```
1.01, a matrix
Done
```

```matrix
amatrix
1.01 1.01
1.01 1.01
```
Fill

**Fill** Value, listVar $\rightarrow$ list

Replaces each element in variable listVar with Value.

*listVar* must already exist.

```
{1,2,3,4,5} $\rightarrow$ alist
{1,2,3,4,5}
```

Fill 1.01,alist Done

```
alist {1.01,1.01,1.01,1.01,1.01}
```

**FiveNumSummary**

**FiveNumSummary** $X\{\text{Freq}\},\{\text{Category}\},\{\text{Include}\}$

Provides an abbreviated version of the 1-variable statistics on list $X$.

A summary of results is stored in the *stat.results* variable. (See page 98.)

$X$ represents a list containing the data.

*Freq* is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding $X$ and *Y* data point. The default value is 1.

*Category* is a list of numeric category codes for the corresponding $X$ data.

*Include* is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists $X$, *Freq*, or *Category* results in a void for the corresponding element of all those lists. For more information on empty elements, see page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.MinX</td>
<td>Minimum of x values.</td>
</tr>
<tr>
<td>stat.Q1X</td>
<td>1st Quartile of x.</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of x.</td>
</tr>
<tr>
<td>stat.Q3X</td>
<td>3rd Quartile of x.</td>
</tr>
<tr>
<td>stat.MaxX</td>
<td>Maximum of x values.</td>
</tr>
</tbody>
</table>

**floor()**

**floor(Value1) $\Rightarrow$ integer**

Returns the greatest integer that is ≤ the argument. This function is identical to *int()*.

The argument can be a real or a complex number.

**floor(List1) $\Rightarrow$ list**

**floor(Matrix1) $\Rightarrow$ matrix**

Returns a list or matrix of the floor of each element.

**Note:** See also *ceiling()* and *int()*.
For

For Var, Low, High [, Step]
Block
EndFor

Executes the statements in Block iteratively for each value of Var, from Low to High, in increments of Step.

Var must not be a system variable.
Step can be positive or negative. The default value is 1.
Block can be either a single statement or a series of statements separated with the “:” character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

format()}

Returns Value as a character string based on the format template.
formatString is a string and must be in the form: "F[n]", "S[n]", "E[n]", "G[n][c]", where [] indicate optional portions.

F[n]: Fixed format. n is the number of digits to display after the decimal point.
S[n]: Scientific format. n is the number of digits to display after the decimal point.
E[n]: Engineering format. n is the number of digits after the first significant digit. The exponent is adjusted to a multiple of three, and the decimal point is moved to the right by zero, one, or two digits.
G[n][c]: Same as fixed format but also separates digits to the left of the radix into groups of three. c specifies the group separator character and defaults to a comma. If c is a period, the radix will be shown as a comma.
[Rc]: Any of the above specifiers may be suffixed with the Rc radix flag, where c is a single character that specifies what to substitute for the radix point.

fPart()}

Returns the fractional part of the argument.
For a list or matrix, returns the fractional parts of the elements.
The argument can be a real or a complex number.

FPdf()}

Computes the F distribution probability at XVal for the specified dfNumer (degrees of freedom) and dfDenom.
**freqTable⟨list()**

`freqTableList(List1, freqIntegerList) \Rightarrow list`

Returns a list containing the elements from `List1` expanded according to the frequencies in `freqIntegerList`. This function can be used for building a frequency table for the Data & Statistics application.

`List1` can be any valid list.

`freqIntegerList` must have the same dimension as `List1` and must contain non-negative integer elements only. Each element specifies the number of times the corresponding `List1` element will be repeated in the result list. A value of zero excludes the corresponding `List1` element.

**Note:** You can insert this function from the computer keyboard by typing `freqTable@>list(...)`. Empty (void) elements are ignored. For more information on empty elements, see page 132.

---

**frequency()**

`frequency(List1, binsList) \Rightarrow list`

Returns a list containing counts of the elements in `List1`. The counts are based on ranges (bins) that you define in `binsList`.

If `binsList` is `{b(1), b(2), ..., b(n)}`, the specified ranges are `{?≤b(1), b(1)<?≤b(2), ... , b(n-1)<?≤b(n), b(n)>?}`. The resulting list is one element longer than `binsList`.

Each element of the result corresponds to the number of elements from `List1` that are in the range of that bin. Expressed in terms of the `countIf()` function, the result is `{ countIf(list, ?≤b(1)), countIf(list, b(1)<?≤b(2)), ..., countIf(list, b(n-1)<?≤b(n)), countIf(list, b(n)>?) }`.

Elements of `List1` that cannot be "placed in a bin" are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 132.

Within the Lists & Spreadsheet application, you can use a range of cells in place of both arguments.

**Note:** See also `countIf()`, page 23.

---

**F Test_2Samp**

`F Test_2Samp List1, List2[, Freq1[, Freq2[, Hypoth]]]]`

`F Test_2Samp List1, List2[, Freq1[, Freq2[, Hypoth]]]]`

(Data list input)

`F Test_2Samp sx1, n1, sx2, n2[, Hypoth]`

`F Test_2Samp sx1, n1, sx2, n2[, Hypoth]`

(Summary stats input)

Performs a two-sample F test. A summary of results is stored in the `stat.results` variable. (See page 98.)

For $H_g: \sigma_1 > \sigma_2$, set `Hypoth=0`

For $H_g: \sigma_1 \neq \sigma_2$ (default), set `Hypoth=0`

For $H_g: \sigma_1 < \sigma_2$, set `Hypoth=0`

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>Calculated F statistic for the data sequence</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.dfNumer</td>
<td>numerator degrees of freedom = n1-1</td>
</tr>
<tr>
<td>stat.dfDenom</td>
<td>denominator degrees of freedom = n2-1</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.x1_bar</td>
<td>Sample means of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>

**Func**

```
Func
    Block
EndFunc
```

Template for creating a user-defined function.

*Block* can be a single statement, a series of statements separated with the ”;” character, or a series of statements on separate lines. The function can use the *Return* instruction to return a specific result.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing [→] instead of [enter] at the end of each line. On the computer keyboard, hold down *Alt* and press *Enter*.

```
Define a piecewise function:
Define g(x)=Func
    If x<0 Then
        Return 3-cos(x)
    Else
        Return 3-x
EndIf
EndFunc
```

**G**

```
gcd()
gcd(Number1, Number2) ⇒ expression
```

Returns the greatest common divisor of the two arguments. The *gcd* of two fractions is the *gcd* of their numerators divided by the *lcm* of their denominators.

In Auto or Approximate mode, the *gcd* of fractional floating-point numbers is 1.0.

```
gcd(List1, List2) ⇒ list
```

Returns the greatest common divisors of the corresponding elements in List1 and List2.

```
gcd(Matrix1, Matrix2) ⇒ matrix
```

Returns the greatest common divisors of the corresponding elements in Matrix1 and Matrix2.

```
gcd(18,33)
gcd(12,14,16,9,7,5)
gcd([2,4],[6,8],[4,12],[8,16])
```

40  **TI-Nspire™ Reference Guide**
**geomCdf()**

`geomCdf(p, lowBound, upBound) ⇒ number if lowBound and upBound are numbers, list if lowBound and upBound are lists
geomCdf(p, upBound) for P(1≤X≤upBound) ⇒ number if upBound is a number, list if upBound is a list

Computes a cumulative geometric probability from lowBound to upBound with the specified probability of success p.

For P(X ≤ upBound), set lowBound = 1.

**geomPdf()**

`geomPdf(p, XVal) ⇒ number if XVal is a number, list if XVal is a list

Computes a probability at XVal, the number of the trial on which the first success occurs, for the discrete geometric distribution with the specified probability of success p.

**getDenom()**

`getDenom(Fraction1) ⇒ value

Transforms the argument into an expression having a reduced common denominator, and then returns its denominator.

\[
\begin{align*}
x &= 5; \quad y = 6 \\
\text{getDenom} &\left(\frac{x+2}{y-3}\right) = 3 \\
\text{getDenom} &\left(\frac{2}{7}\right) = 7 \\
\text{getDenom} &\left(\frac{1}{x} + \frac{y^2 + y}{y^2}\right) = 30
\end{align*}
\]

**getLangInfo()**

`getLangInfo() ⇒ string

Returns a string that corresponds to the short name of the currently active language. You can, for example, use it in a program or function to determine the current language.

- English = "en"
- Danish = "da"
- German = "de"
- Finnish = "fi"
- French = "fr"
- Italian = "it"
- Dutch = "nl"
- Belgian Dutch = "nl_BE"
- Norwegian = "no"
- Portuguese = "pt"
- Spanish = "es"
- Swedish = "sv"
getLockInfo() \(\Rightarrow\) value

Returns the current locked/unlocked state of variable \(\text{Var}\).

- \(\text{value} = 0\): \(\text{Var}\) is unlocked or does not exist.
- \(\text{value} = 1\): \(\text{Var}\) is locked and cannot be modified or deleted.

See Lock, page 57, and unlock, page 110.

getMode() \(\Rightarrow\) value

- \(\text{getMode(0)}\) \(\Rightarrow\) list

getMode(\(\text{ModeNameInteger}\)) returns a value representing the current setting of the \(\text{ModeNameInteger}\) mode.

getMode(0) returns a list containing number pairs. Each pair consists of a mode integer and a setting integer.

For a listing of the modes and their settings, refer to the table below.

If you save the settings with \(\text{getMode(0)} \Rightarrow \text{var}\), you can use \(\text{setMode(\text{var})}\) in a function or program to temporarily restore the settings within the execution of the function or program only. See setMode(), page 91.

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td>1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12, 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12</td>
</tr>
<tr>
<td>Angle</td>
<td>2</td>
<td>1=Radian, 2=Degree, 3=Gradian</td>
</tr>
<tr>
<td>Exponential Format</td>
<td>3</td>
<td>1=Normal, 2=Scientific, 3=Engineering</td>
</tr>
<tr>
<td>Real or Complex</td>
<td>4</td>
<td>1=Real, 2=Rectangular, 3=Polar</td>
</tr>
<tr>
<td>Auto or Approx.</td>
<td>5</td>
<td>1=Auto, 2=Approximate</td>
</tr>
<tr>
<td>Vector Format</td>
<td>6</td>
<td>1=Rectangular, 2=Cylindrical, 3=Spherical</td>
</tr>
<tr>
<td>Base</td>
<td>7</td>
<td>1=Decimal, 2=Hex, 3=Binary</td>
</tr>
</tbody>
</table>
getNum()

getNum(Fraction1) \Rightarrow value

Transforms the argument into an expression having a reduced common denominator, and then returns its numerator.

\[
\begin{align*}
x&=5; \ y=6 \\
\text{getNum}\left(\frac{x+2}{y-3}\right) &\Rightarrow 7 \\
\text{getNum}\left(\frac{2}{7}\right) &\Rightarrow 2 \\
\text{getNum}\left(\frac{1}{x} + \frac{1}{y}\right) &\Rightarrow 11
\end{align*}
\]

g getType()

g getType(var) \Rightarrow string

Returns a string that indicates the data type of variable var. If var has not been defined, returns the string "NONE".

\[
\begin{align*}
\{1,2,3\} &\rightarrow \text{temp} \\
\text{getType}(&\text{temp}) &\Rightarrow "LIST" \\
2 \cdot 3 \cdot t &\rightarrow \text{temp} \\
\text{getType}(&\text{temp}) &\Rightarrow "EXPR" \\
\text{DelVar} &\text{temp} \\
\text{getType}(&\text{temp}) &\Rightarrow "NONE"
\end{align*}
\]

g getVarInfo()

g getVarInfo() \Rightarrow matrix or string

g getVarInfo(LibNameString) \Rightarrow matrix or string

getVarInfo() returns a matrix of information (variable name, type, library accessibility, and locked/unlocked state) for all variables and library objects defined in the current problem. If no variables are defined, getVarInfo() returns the string "NONE".

\[
\begin{align*}
\text{getVarInfo}() &\Rightarrow \text{"NONE"} \\
\text{Define } x=5 &\text{ Done} \\
\text{Lock } x &\text{ Done} \\
\text{Define LibPriv.y=\{1,2,3\}} &\text{ Done} \\
\text{Define LibPub.} &\text{ Done}
\end{align*}
\]

getVarInfo(LibNameString) returns a matrix of information for all library objects defined in library LibNameString. LibNameString must be a string (text enclosed in quotation marks) or a string variable.

If the library LibNameString does not exist, an error occurs.

\[
\begin{align*}
\text{getVarInfo}() &\Rightarrow \begin{bmatrix} x & \text{"NUM"} & \{2\} & 1 \\
y & \text{"LIST"} & \text{"LibPriv"} & 0 \\
z & \text{"FUNC"} & \text{"LibPub"} & 0 \end{bmatrix} \\
\text{getVarInfo}(\text{"tmp3"}) &\Rightarrow \text{"Error: Argument must be a string"} \\
\text{getVarInfo}(\text{"tmp3"}) &\Rightarrow \begin{bmatrix} \text{volcy2} & \text{"NONE"} & \text{"LibPub"} & 0 \end{bmatrix}
\end{align*}
\]
Note the example, in which the result of `getVarInfo()` is assigned to variable `vs`. Attempting to display row 2 or row 3 of `vs` returns an "Invalid list or matrix" error because at least one of elements in those rows (variable `b`, for example) reevaluates to a matrix.

This error could also occur when using `Ans` to reevaluate a `getVarInfo()` result.

The system gives the above error because the current version of the software does not support a generalized matrix structure where an element of a matrix can be either a matrix or a list.

---

**Goto**

**Goto** `labelName`

Transfers control to the label `labelName`.

`labelName` must be defined in the same function using a `Lbl` instruction.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `[]` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

---

**Grad**

`Expr1 › Grad`  `→`  `expression`

Converts `Expr1` to gradian angle measure.

**Note:** You can insert this operator from the computer keyboard by typing `@>Grad`.

---

In Degree angle mode:

```
(1.5 › Grad)  \rightarrow  1.66667^g
```

In Radian angle mode:

```
(1.5 › Grad)  \rightarrow  (95.493)^g
```
**identity()**

`identity(Integer) → matrix`

Returns the identity matrix with a dimension of `Integer`.

`Integer` must be a positive integer.

<table>
<thead>
<tr>
<th><code>identity(4)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

**If**

- **If** `BooleanExpr` **Then**
  - **Statement**
  - **EndIf**

If `BooleanExpr` evaluates to true, executes the single statement `Statement` or the block of statements `Block` before continuing execution.

If `BooleanExpr` evaluates to false, continues execution without executing the statement or block of statements.

`Block` can be either a single statement or a sequence of statements separated with the `:` character.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `↵` instead of `↵` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

- **If** `BooleanExpr` **Then**
  - **Block1**
- **Else**
  - **Block2**
- **EndIf**

If `BooleanExpr` evaluates to true, executes `Block1` and then skips `Block2`. If `BooleanExpr` evaluates to false, skips `Block1` but executes `Block2`.

`Block1` and `Block2` can be a single statement.

Define \( g(x) = \text{Func} \)

- If \( x < 0 \) Then
  - Return \( x^2 \)
  - EndFunc

\( g(-2) \)

Done

12

Define \( g(x) = \text{Func} \)

- If \( x < 0 \) Then
  - Return \( -x \)
- Else
  - Return \( x \)
  - EndIf
  - EndFunc

\( g(12) \)

\( g(-12) \)

Done

12

12
If

If BooleanExpr1 Then
  Block1
ElseIf BooleanExpr2 Then
  Block2
  ...
ElseIf BooleanExprN Then
  BlockN
EndIf

Allows for branching. If BooleanExpr1 evaluates to true, executes Block1. If BooleanExpr1 evaluates to false, evaluates BooleanExpr2, and so on.

Define \( g(x) = \) Func

- If \( x < 5 \) Then
  Return 5
- ElseIf \( x > 5 \) and \( x < 0 \) Then
  Return \( x \)
- ElseIf \( x \geq 0 \) and \( x + 10 \) Then
  Return \( x \)
- ElseIf \( x = 10 \) Then
  Return 3
EndIf
EndFunc

\( g(-4) \)
\[ g(-4) = 4 \]

\( g(10) \)
\[ g(10) = 3 \]

\( \text{imag}() \)

\( \text{imag}(\text{Value1}) \Rightarrow \text{value} \)

Returns the imaginary part of the argument.

\( \text{imag}(\text{List1}) \Rightarrow \text{list} \)

Returns a list of the imaginary parts of the elements.

\( \text{imag}(1+2\cdot i) \)
\[ \text{imag}(1+2\cdot i) = 2 \]

\( \text{imag}\{[-3,4-i,i]\} \)
\[ \text{imag}\{[-3,4-i,i]\} = \{0,1,1\} \]
imag()

imag\[Matrix\] \Rightarrow matrix

Returns a matrix of the imaginary parts of the elements.

\[
\begin{pmatrix}
1 & 2 \\
i \cdot 3 & i \cdot 4
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
0 & 0 \\
3 & 4
\end{pmatrix}
\]

Indirection

See #(), page 127.

inString()

\text{inString}(srcString, subString[, Start]) \Rightarrow integer

Returns the character position in string \text{srcString} at which the first occurrence of string \text{subString} begins.

\text{Start}, if included, specifies the character position within \text{srcString} where the search begins. Default = 1 (the first character of \text{srcString}).

If \text{srcString} does not contain \text{subString} or \text{Start} is > the length of \text{srcString}, returns zero.

\[
\begin{align*}
\text{inString}("Hello there","the") & \Rightarrow 7 \\
\text{inString}("ABCEFG","D") & \Rightarrow 0
\end{align*}
\]

int()

\text{int}(Value) \Rightarrow integer
\text{int}(List) \Rightarrow list
\text{int}(Matrix) \Rightarrow matrix

Returns the greatest integer that is less than or equal to the argument. This function is identical to \text{floor()}.

The argument can be a real or a complex number.

For a list or matrix, returns the greatest integer of each of the elements.

\[
\begin{align*}
\text{int}(-2.5) & \Rightarrow -3. \\
\text{int}([-1.234 \ 0.37]) & \Rightarrow [\ -2. \ 0 \ 0. \ ]
\end{align*}
\]

intDiv()

\text{intDiv}(Number1, Number2) \Rightarrow integer
\text{intDiv}(List1, List2) \Rightarrow list
\text{intDiv}(Matrix1, Matrix2) \Rightarrow matrix

Returns the signed integer part of \((Number1 \div Number2)\).

For lists and matrices, returns the signed integer part of \((\text{argument 1} \div \text{argument 2})\) for each element pair.

\[
\begin{align*}
\text{intDiv}(-7,2) & \Rightarrow -3 \\
\text{intDiv}(4,5) & \Rightarrow 0 \\
\text{intDiv}\{\{-12\,-14\,-16\}\,\{5,4,3\}\} & \Rightarrow \{\ 2, \ 3, \ 5\ \}
\end{align*}
\]
interpolate

```
interpolate(xValue, xList, yList, yPrimeList) \Rightarrow list
```

This function does the following:

Given `xList`, `yList=f(xList)`, and `yPrimeList=f'(xList)` for some unknown function `f`, a cubic interpolant is used to approximate the function `f` at `xValue`. It is assumed that `xList` is a list of monotonically increasing or decreasing numbers, but this function may return a value even when it is not. This function walks through `xList` looking for an interval `[xList[i], xList[i+1]]` that contains `xValue`. If it finds such an interval, it returns an interpolated value for `f(xValue)`; otherwise, it returns `undef`.

`xList`, `yList, and `yPrimeList` must be of equal dimension ≥ 2 and contain expressions that simplify to numbers.
`xValue` can be a number or a list of numbers.

Differential equation:
```
y'=-3·y+6·t+5 and y(0)=5
```

```plaintext
rk=rk23\{-3·y+6·t+5f'(xList),(0.10),(5.1)
0. 1. 2. 3. 4.
5. 3.19499 5.00394 6.99957 9.00593 10
```

To see the entire result, press `\` and then use `\` and `\` to move the cursor.

Use the interpolate() function to calculate the function values for the `xvaluelist`:

```plaintext
xvaluelist:=seq{i,i,0,10,0.5)
\{0,0.5,1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,
\}
```
```plaintext
xlist:=mat\[list(rk[1]
\}
```
```plaintext
ylist:=mat\[list(rk[2]
\{5.,3.19499,5.00394,6.99957,9.00593,10.997
\}
```
```plaintext
yprimeList:=-3·y+6·t+5 and t=xlist
\{10.,1.41503,1.98819,2.00129,1.98221,2.00
```
```plaintext
interpolate(xvaluelist,xlist,ylist,yprimeList)
\{5.,2.67062,3.19499,4.02782,5.00394,6.0001
```

-inv\(^2\)

```
inv\(\chi^2\)(Area,df)
invChi2(Area,df)
```

Computes the Inverse cumulative \(\chi^2\) (chi-square) probability function specified by degree of freedom, `df` for a given `Area` under the curve.

-inv\(F\)

```
inv\(F\)(Area,df Numer,df Denom)
invF(Area,df Numer,df Denom)
```

Computes the Inverse cumulative \(F\) distribution function specified by `df Numer` and `df Denom` for a given `Area` under the curve.

-invNorm()

```
invNorm(Area,\mu,\sigma)
```

Computes the inverse cumulative normal distribution function for a given `Area` under the normal distribution curve specified by `\mu` and `\sigma`.

-inv\(t\)

```
invt(Area,df)
```

Computes the inverse cumulative Student-t probability function specified by degree of freedom, `df` for a given `Area` under the curve.
iPart()  

\[ \text{iPart}(\text{Number}) \Rightarrow \text{integer} \]  
\[ \text{iPart}(\text{List1}) \Rightarrow \text{list} \]  
\[ \text{iPart}(\text{Matrix1}) \Rightarrow \text{matrix} \]  

Returns the integer part of the argument.  

For lists and matrices, returns the integer part of each element.  

The argument can be a real or a complex number.

\[ \text{iPart}(1.234) = -1 \]  
\[ \text{iPart}\left(\frac{3}{2},-2,3,7.003\right) = \{1,2,7\} \]

irr()  

\[ \text{irr}(\text{CF0},\text{CFList},[\text{CFFreq}]) \Rightarrow \text{value} \]  

Financial function that calculates internal rate of return of an investment.  

\(\text{CF0}\) is the initial cash flow at time 0; it must be a real number.  

\(\text{CFList}\) is a list of cash flow amounts after the initial cash flow \(\text{CF0}\).  

\(\text{CFFreq}\) is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of \(\text{CFList}\). The default is 1; if you enter values, they must be positive integers < 10,000.  

Note: See also \text{mirr}(), page 63.

isPrime()  

\[ \text{isPrime}(\text{Number}) \Rightarrow \text{Boolean constant expression} \]  

Returns true or false to indicate if \(\text{number}\) is a whole number \(\geq 2\) that is evenly divisible only by itself and 1.  

If \(\text{Number}\) exceeds about 306 digits and has no factors \(\leq 1021\), \text{isPrime}(\text{Number}) displays an error message.  

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \(\text{\textbf{\textendash}}\) instead of \(\text{\textbf{\textendash}}\) at the end of each line. On the computer keyboard, hold down \(\text{\textbf{Alt}}\) and press \(\text{\textbf{Enter}}\).

\text{isPrime}(5) = \text{true} \]  
\text{isPrime}(6) = \text{false} \]

isVoid()  

\[ \text{isVoid}(\text{Var}) \Rightarrow \text{Boolean constant expression} \]  
\[ \text{isVoid}(\text{Expr}) \Rightarrow \text{Boolean constant expression} \]  
\[ \text{isVoid}(\text{List}) \Rightarrow \text{list of Boolean constant expressions} \]  

Returns true or false to indicate if the argument is a void data type.  

For more information on void elements, see page 132.

\[ \text{isVoid}(a) = \text{true} \]  
\[ \text{isVoid}\left(\{1,\_3\}\right) = \{\text{false, true, false}\} \]
**Lbl**

**Lbl** labelName

Defines a label with the name `labelName` within a function.

You can use a `Goto labelName` instruction to transfer control to the instruction immediately following the label.

`labelName` must meet the same naming requirements as a variable name.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `↵` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

```
Define g()=Func
  Local temp,i
  0→temp
  1→i
  Lbl top
  temp+i→temp
  If i<10 Then
    i+1→i
    Goto top
  EndIf
  Return temp
EndFunc
```

---

**lcm()**

**lcm(Number1, Number2) ⇒ expression**

**lcm(List1, List2) ⇒ list**

**lcm(Matrix1, Matrix2) ⇒ matrix**

Returns the least common multiple of the two arguments. The `lcm` of two fractions is the `lcm` of their numerators divided by the `gcd` of their denominators. The `lcm` of fractional floating-point numbers is their product.

For two lists or matrices, returns the least common multiples of the corresponding elements.

```
{lcm(6,9)}
```

---

**left()**

**left(sourceString, Num) ⇒ string**

Returns the leftmost `Num` characters contained in character string `sourceString`.

If you omit `Num`, returns all of `sourceString`.

**left(List1, Num) ⇒ list**

Returns the leftmost `Num` elements contained in `List1`.

If you omit `Num`, returns all of `List1`.

**left(Comparison) ⇒ expression**

Returns the left-hand side of an equation or inequality.

```
{left("Hello",2)}
```

---
libShortcut

libShortcut([LibNameString, ShortcutNameString [, LibPrivFlag]]) → list of variables

Creates a variable group in the current problem that contains references to all the objects in the specified library document LibNameString. Also adds the group members to the Variables menu. You can then refer to each object using its ShortcutNameString.

Set LibPrivFlag=0 to exclude private library objects (default).
Set LibPrivFlag=1 to include private library objects.

To copy a variable group, see CopyVar on page 18.
To delete a variable group, see DelVar on page 29.

LinRegBx

LinRegBx X,Y,[Freq],[Category,Include])

Computes the linear regression \( y = a + b \cdot x \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

All the lists must have equal dimension except for \( \text{Include} \).
\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Category \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression Equation: ( a + b \cdot x )</td>
</tr>
<tr>
<td>( \text{stat.a, stat.b} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.r}^2 )</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>( \text{stat.r} )</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>( \text{stat.Resid} )</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>( \text{stat.XReg} )</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq ), ( Category ) List, and ( Include ) Categories</td>
</tr>
<tr>
<td>( \text{stat.YReg} )</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq ), ( Category ) List, and ( Include ) Categories</td>
</tr>
<tr>
<td>( \text{stat.FreqReg} )</td>
<td>List of frequencies corresponding to ( \text{stat.XReg} ) and ( \text{stat.YReg} )</td>
</tr>
</tbody>
</table>
LinRegMx  \(X,Y,[\text{Freq}],[\text{Category},\text{Include}])\)

Computes the linear regression \(y = m \cdot x + b\) on lists \(X\) and \(Y\) with frequency \(\text{Freq}\). A summary of results is stored in the \(\text{stat.results}\) variable. (See page 98.)

All the lists must have equal dimension except for \(\text{Include}\).

\(X\) and \(Y\) are lists of independent and dependent variables.

\(\text{Freq}\) is an optional list of frequency values. Each element in \(\text{Freq}\) specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

\(\text{Category}\) is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\(\text{Include}\) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: (y = m \cdot x + b)</td>
</tr>
<tr>
<td>stat.m, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified (X) List actually used in the regression based on restrictions of (\text{Freq}, \text{Category List}, \text{and Include Categories})</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified (Y) List actually used in the regression based on restrictions of (\text{Freq}, \text{Category List}, \text{and Include Categories})</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to (\text{stat.XReg}) and (\text{stat.YReg})</td>
</tr>
</tbody>
</table>

LinRegtIntervals  \(X,Y,[\text{Freq}],[0],[\text{CLev}])\]

For Slope. Computes a level \(C\) confidence interval for the slope.

LinRegtIntervals  \(X,Y,[\text{Freq}],[1],[\text{Xval}],[\text{CLev}])\]

For Response. Computes a predicted \(y\)-value, a level \(C\) prediction interval for a single observation, and a level \(C\) confidence interval for the mean response.

A summary of results is stored in the \(\text{stat.results}\) variable. (See page 98.)

All the lists must have equal dimension.

\(X\) and \(Y\) are lists of independent and dependent variables.

\(F\) is an optional list of frequency values. Each element in \(F\) specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.
### Output variable | Description
--- | ---
stat.RegEqn | Regression Equation: \( a + b \cdot x \)
stat.a, stat.b | Regression coefficients
stat.df | Degrees of freedom
stat.\( r^2 \) | Coefficient of determination
stat.r | Correlation coefficient
stat.Resid | Residuals from the regression

#### For Slope type only

### Output variable | Description
--- | ---
[stat.CLower, stat.CUpper] | Confidence interval for the slope
stat.ME | Confidence interval margin of error
stat.SEslope | Standard error of slope
stat.s | Standard error about the line

#### For Response type only

### Output variable | Description
--- | ---
[stat.CLower, stat.CUpper] | Confidence interval for the mean response
stat.ME | Confidence interval margin of error
stat.SE | Standard error of mean response
[stat.LowerPred, stat.UpperPred] | Prediction interval for a single observation
stat.MEPred | Prediction interval margin of error
stat.SEPred | Standard error for prediction
stat.\( \hat{y} \) | \( a + b \cdot XVal \)
LinRegTTest \( X, Y, Freq[, Hypoth] \)

Computes a linear regression on the \( X \) and \( Y \) lists and a \( t \) test on the value of slope \( \beta \) and the correlation coefficient \( \rho \) for the equation \( y = a + \beta x \). It tests the null hypothesis \( H_0: \beta = 0 \) (equivalently, \( \rho = 0 \)) against one of three alternative hypotheses.

All the lists must have equal dimension.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Hypoth \) is an optional value specifying one of three alternative hypotheses against which the null hypothesis \( (H_0: \beta = \rho = 0) \) will be tested.

For \( H_a: \beta \neq 0 \) and \( \rho \neq 0 \) (default), set \( Hypoth = 0 \)
For \( H_a: \beta < 0 \) and \( \rho < 0 \), set \( Hypoth < 0 \)
For \( H_a: \beta > 0 \) and \( \rho > 0 \), set \( Hypoth > 0 \)

A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

### Output variable | Description
--- | ---
\( \text{stat.RegEqn} \) | Regression equation: \( a + b \cdot x \)
\( \text{stat.t} \) | \( t \)-Statistic for significance test
\( \text{stat.PVal} \) | Smallest level of significance at which the null hypothesis can be rejected
\( \text{stat.df} \) | Degrees of freedom
\( \text{stat.a}, \text{stat.b} \) | Regression coefficients
\( \text{stat.s} \) | Standard error about the line
\( \text{stat.SESlope} \) | Standard error of slope
\( \text{stat.}^2 \) | Coefficient of determination
\( \text{stat.}r \) | Correlation coefficient
\( \text{stat.Resid} \) | Residuals from the regression
linSolve()

linSolve(SystemOfLinearEqns, Var1, Var2, ...) ⇒ list
linSolve(LinearEqn1 and LinearEqn2 and ..., Var1, Var2, ...) ⇒ list
linSolve((LinearEqn1, LinearEqn2, ...), Var1, Var2, ...) ⇒ list
linSolve(SystemOfLinearEqns, {Var1, Var2, ...}) ⇒ list
linSolve(LinearEqn1 and LinearEqn2 and ..., {Var1, Var2, ...}) ⇒ list
linSolve((LinearEqn1, LinearEqn2, ...), {Var1, Var2, ...}) ⇒ list

Returns a list of solutions for the variables Var1, Var2, ...

The first argument must evaluate to a system of linear equations or a single linear equation. Otherwise, an argument error occurs.

For example, evaluating \( \text{linSolve}(x=1 \text{ and } x=2, x) \) produces an “Argument Error” result.

\[ \begin{align*}
\text{linSolve} & \left( \begin{array}{c}
2x+y=3 \\
5x-3y=7
\end{array} \right), \{x,y\} \\
\text{linSolve} & \left( \begin{array}{c}
2x=3 \\
5x-3y=7
\end{array} \right), \{x,y\} \\
\text{linSolve} & \left( \begin{array}{c}
apple+4\text{pear}=23 \\
5\text{apple}-\text{pear}=17
\end{array} \right), \{\text{apple,pear}\} \\
\text{linSolve} & \left( \begin{array}{c}
apple\cdot4\text{pear}=\frac{14}{3} \\
-\text{apple}+\text{pear}=6
\end{array} \right), \{\text{apple,pear}\}
\end{align*} \]

\[ \begin{align*}
\text{linSolve} & \left( \begin{array}{c}
apple+4\text{pear}=23 \\
5\text{apple}-\text{pear}=17
\end{array} \right), \{\text{apple,pear}\} \\
\text{linSolve} & \left( \begin{array}{c}
apple\cdot4\text{pear}=\frac{14}{3} \\
-\text{apple}+\text{pear}=6
\end{array} \right), \{\text{apple,pear}\}
\end{align*} \]

\[ \begin{align*}
\Delta\text{List} & \left( \begin{array}{c}
20, 30, 45, 70
\end{array} \right) \\
\Delta\text{List} & \left( \begin{array}{c}
\text{apple,pear}
\end{array} \right)
\end{align*} \]

\[ \begin{align*}
\Delta\text{List} & \left( \begin{array}{c}
20, 30, 45, 70
\end{array} \right) \\
\Delta\text{List} & \left( \begin{array}{c}
\text{apple,pear}
\end{array} \right)
\end{align*} \]

\[ \begin{align*}
\text{list\texttt{mat}} & \left( \begin{array}{c}
1, 2, 3
\end{array} \right) \\
\text{list\texttt{mat}} & \left( \begin{array}{c}
1, 2, 3, 4, 5
\end{array} \right), 2
\end{align*} \]

\[ \begin{align*}
\text{list\texttt{mat}} & \left( \begin{array}{c}
1, 2, 3
\end{array} \right) \\
\text{list\texttt{mat}} & \left( \begin{array}{c}
1, 2, 3, 4, 5
\end{array} \right), 2
\end{align*} \]

\[ \begin{align*}
\ln(2) & \text{ produces } 0.693147
\end{align*} \]

\[ \begin{align*}
\ln(3,1,2,5) & \text{ produces } "\text{Error: Non–real calculation}" \\
\ln(-3,1,2,5) & \{1.09861+3.14159\text{i},0.182322,1.60944\}
\end{align*} \]
ln() \[ \ln(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

Returns the matrix natural logarithm of squareMatrix1. This is not the same as calculating the natural logarithm of each element. For information about the calculation method, refer to \( \cos() \) on.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

\[
\begin{pmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{pmatrix}
\]

\[
\begin{align*}
\ln & 1.83145+1.73485\cdot i \\
& 0.448761-0.72553\cdot i \\
& -0.266891-2.08316\cdot i
\end{align*}
\]

To see the entire result, press $\text{\textasciicircum}$ and then use $\text{\textasciicircum}$ and $\text{\textasciicircum}$ to move the cursor.

\( \text{LnReg} \)

\( \text{LnReg} X, Y, [\text{Freq}] [, \text{Category}, \text{Include}] \)

Computes the logarithmic regression \( y = a+b \cdot \ln(x) \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 98.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression equation: ( a+b \cdot \ln(x) )</td>
</tr>
<tr>
<td>( \text{stat.a, stat.b} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.}^2 )</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>( \text{stat.r} )</td>
<td>Correlation coefficient for transformed data (( \ln(x), y ))</td>
</tr>
<tr>
<td>( \text{stat.Resid} )</td>
<td>Residuals associated with the logarithmic model</td>
</tr>
<tr>
<td>( \text{stat.ResidTrans} )</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>( \text{stat.XReg} )</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( \text{Freq}, \ \text{Category List}, \ \text{and Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.YReg} )</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( \text{Freq}, \ \text{Category List}, \ \text{and Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.FreqReg} )</td>
<td>List of frequencies corresponding to ( \text{stat.XReg} ) and ( \text{stat.YReg} )</td>
</tr>
</tbody>
</table>
Local

Local Var1[, Var2] [, Var3] ...

Declares the specified vars as local variables. Those variables exist only during evaluation of a function and are deleted when the function finishes execution.

Note: Local variables save memory because they only exist temporarily. Also, they do not disturb any existing global variable values. Local variables must be used for For loops and for temporarily saving values in a multi-line function since modifications on global variables are not allowed in a function.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \( \text{Enter} \) instead of \( \text{Enter} \) at the end of each line. On the computer keyboard, hold down Alt and press Enter.

```
Define rollcount()=Func
    Local i
    1 → i
    Loop
    If randInt(1,6)=randInt(1,6)
    Goto end
    i+1 → i
    EndLoop
    Lbl end
    Return i
EndFunc
```

Lock

Lock Var1[, Var2] [, Var3] ...

Lock Var.

Locks the specified variables or variable group. Locked variables cannot be modified or deleted.

You cannot lock or unlock the system variable Ans, and you cannot lock the system variable groups stat or tvm.

Note: The Lock command clears the Undo/Redo history when applied to unlocked variables.

See unlock, page 110, and getLockInfo(), page 42.

```
a:=65
Lock a
Done
getLockInfo(a) 1
a:=75 "Error: Variable is locked."
DelVar a "Error: Variable is locked."
Unlock a  Done
a:=75 75
DelVar a  Done
```
\textbf{log()}

\begin{align*}
\text{log}(\text{Value1}, \text{Value2}) & \Rightarrow \text{value} \\
\text{log}(\text{List1}, \text{Value2}) & \Rightarrow \text{list}
\end{align*}

Returns the base-Value2 logarithm of the first argument.

**Note:** See also \textbf{Log template}, page 2.

For a list, returns the base-Value2 logarithm of the elements.

If the second argument is omitted, 10 is used as the base.

\[
\begin{array}{c|c}
\text{log} & \text{value} \\
\hline
\log_{10}(2.) & 0.30103 \\
\log_4(2.) & 0.5 \\
\frac{\log_{10}(10) - \log_{10}(5)}{3} & 0.63093
\end{array}
\]

If complex format mode is Real:

\[
\log_{10}\left\{-3,1,2,5\right\}
\]

"Error: Non–real calculation"

If complex format mode is Rectangular:

\[
\log_{10}\left\{-3,1,2,5\right\} = \left\{0.477121+1.36438\cdot i, 0.079181, 0.69897\right\}
\]

\textbf{log(squareMatrix1, Value2) \Rightarrow squareMatrix}

Returns the matrix base-Value logarithm of squareMatrix1. This is not the same as calculating the base-Value logarithm of each element. For information about the calculation method, refer to \textbf{cos()}.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

If the base argument is omitted, 10 is used as base.

\[
\begin{array}{c|c}
\text{log} & \\
\hline
\log_{10}\left[ \begin{array}{ccc} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{array} \right] & \left[0.795387+0.753438\cdot i, 0.003993-0.6474\cdot i, 0.194895-0.315095\cdot i, 0.462485+0.2707\cdot i, 0.159590-0.0904706\cdot i, 0.488304+0.7774\cdot i\right]
\end{array}
\]

To see the entire result, press \(\uparrow\) and then use \(\downarrow\) and \(\rightarrow\) to move the cursor.

\textbf{Logistic}

\textbf{Logistic} \(X, Y[, \text{Freq}], [\text{Category, Include}]\)

Computes the logistic regression \(y = \frac{c}{1+a \cdot e^{-bx}}\) on lists \(X\) and \(Y\) with frequency \(\text{Freq}\). A summary of results is stored in the \text{stat.results} variable. (See page 98.)

All the lists must have equal dimension except for \text{Include}.

\(X\) and \(Y\) are lists of independent and dependent variables.

\(\text{Freq}\) is an optional list of frequency values. Each element in \(\text{Freq}\) specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

\text{Category} is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\text{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.
### LogisticD

**LogisticD** \( X, Y [, \text{Iterations}] [, \text{Freq}] [, \text{Category}, \text{Include}] \)

Computes the logistic regression \( y = \frac{c}{1 + a \cdot e^{-bX}} + d \) on lists \( X \) and \( Y \) with frequency \( Freq \), using a specified number of \text{Iterations}. A summary of results is stored in the \text{stat.results} variable. (See page 98.)

All the lists must have equal dimension except for \text{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\text{Freq} is an optional list of frequency values. Each element in \text{Freq} specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\text{Category} is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\text{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.RegEqn}</td>
<td>Regression equation: ( \frac{c}{1 + a \cdot e^{-bX}} + d )</td>
</tr>
<tr>
<td>\text{stat.a}, \text{stat.b}, \text{stat.c}, \text{stat.d}</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>\text{stat.Resid}</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>\text{stat.XReg}</td>
<td>List of data points in the modified \text{X List} actually used in the regression based on restrictions of \text{Freq}, \text{Category List}, and \text{Include Categories}</td>
</tr>
<tr>
<td>\text{stat.YReg}</td>
<td>List of data points in the modified \text{Y List} actually used in the regression based on restrictions of \text{Freq}, \text{Category List}, and \text{Include Categories}</td>
</tr>
<tr>
<td>\text{stat.FreqReg}</td>
<td>List of frequencies corresponding to \text{stat.XReg} and \text{stat.YReg}</td>
</tr>
</tbody>
</table>
Loop

**Block**

**EndLoop**

Repeatedly executes the statements in **Block**. Note that the loop will be executed endlessly, unless a **Goto** or **Exit** instruction is executed within **Block**.

**Block** is a sequence of statements separated with the “:” character.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \[\text{alt} \text{enter} \] at the end of each line. On the computer keyboard, hold down \[\text{alt} \] and press \[\text{enter} \].

---

**LU**

**LU** **Matrix, lMatrix, uMatrix, pMatrix, Tol**

Calculates the Doolittle LU (lower-upper) decomposition of a real or complex matrix. The lower triangular matrix is stored in **lMatrix**, the upper triangular matrix in **uMatrix**, and the permutation matrix (which describes the row swaps done during the calculation) in **pMatrix**.

**lMatrix · uMatrix = pMatrix · matrix**

Optionally, any matrix element is treated as zero if its absolute value is less than **Tol**. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, **Tol** is ignored.

- If you use \[\text{ctrl enter} \] or set the **Auto** or **Approximate** mode to **Approximate**, computations are done using floating-point arithmetic.
- If **Tol** is omitted or not used, the default tolerance is calculated as:
  \[5 \times 10^{-14} \times \text{max(dim} (\text{Matrix})) \times \text{rowNorm} (\text{Matrix})\]

The **LU** factorization algorithm uses partial pivoting with row interchanges.

---

**M**

**mat\#list()**

**mat\#list** **(Matrix) \(\Rightarrow\) list**

Returns a list filled with the elements in **Matrix**. The elements are copied from **Matrix** row by row.

**Note:** You can insert this function from the computer keyboard by typing \[\text{mat}\#\text{list} (\ldots)\].
max()

\[
\text{max}(\text{Value}_1, \text{Value}_2) \Rightarrow \text{expression}
\]
\[
\text{max}([\text{List}_1, \text{List}_2]) \Rightarrow \text{list}
\]
\[
\text{max}([\text{Matrix}_1, \text{Matrix}_2]) \Rightarrow \text{matrix}
\]

Returns the maximum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the maximum value of each pair of corresponding elements.

\[
\text{max}([\text{List}]) \Rightarrow \text{expression}
\]

Returns the maximum element in list.

\[
\text{max}([\text{Matrix}_1]) \Rightarrow \text{matrix}
\]

Returns a row vector containing the maximum element of each column in Matrix1.

Empty (void) elements are ignored. For more information on empty elements, see page 132.

**Note:** See also \texttt{min()}.

mean()

\[
\text{mean}([\text{List}_1, \text{freqList}_1]) \Rightarrow \text{expression}
\]

Returns the mean of the elements in List. Each \texttt{freqList} element counts the number of consecutive occurrences of the corresponding element in List.

\[
\text{mean}([\text{Matrix}_1, \text{freqMatrix}_1]) \Rightarrow \text{matrix}
\]

Returns a row vector of the means of all the columns in Matrix1. Each \texttt{freqMatrix} element counts the number of consecutive occurrences of the corresponding element in Matrix1.

Empty (void) elements are ignored. For more information on empty elements, see page 132.

\[
\text{mean}([\text{List}]) \Rightarrow \text{expression}
\]

\[
\text{mean}([\text{Matrix}_1]) \Rightarrow \text{matrix}
\]

In Rectangular vector format:

\[
\text{mean}\left(\begin{bmatrix}
0.2 & 0 & 0.1 & 0.3 & 0.4 \\
-1 & 3 & -0.13333 & 0.833333 \\
0.4 & -0.5
\end{bmatrix}\right)
\]

\[
\text{mean}\left(\begin{bmatrix}
1 & 0 & -2 & 5 \\
5 & 3 & 15 & 6
\end{bmatrix}\right)
\]

\[
\text{mean}\left(\begin{bmatrix}
1 & 2 & 5 & 3 \\
3 & 4 & 4 & 1 \\
5 & 6 & 6 & 2
\end{bmatrix}\right)
\]

median()

\[
\text{median}([\text{List}_1, \text{freqList}_1]) \Rightarrow \text{expression}
\]

Returns the median of the elements in List. Each \texttt{freqList} element counts the number of consecutive occurrences of the corresponding element in List.

\[
\text{median}([\text{List}]) \Rightarrow \text{expression}
\]

\[
\text{median}([\text{Matrix}_1]) \Rightarrow \text{matrix}
\]
### median()

**median(\(\text{Matrix1}, \text{freqMatrix}\)) \Rightarrow \text{matrix}**

Returns a row vector containing the medians of the columns in \(\text{Matrix1}\). Each \(\text{freqMatrix}\) element counts the number of consecutive occurrences of the corresponding element in \(\text{Matrix1}\).

**Notes:**
- All entries in the list or matrix must simplify to numbers.
- Empty (void) elements in the list or matrix are ignored. For more information on empty elements, see page 132.

### MedMed

**MedMed \(X, Y [, Freq [, \text{Category}, \text{Include}]\)**

Computes the median-median line \(y = (m \cdot x+b)\) on lists \(X\) and \(Y\) with frequency \(Freq\). A summary of results is stored in the \(\text{stat.results}\) variable. (See page 98.)

All the lists must have equal dimension except for \(\text{Include}\).

\(X\) and \(Y\) are lists of independent and dependent variables.

\(Freq\) is an optional list of frequency values. Each element in \(Freq\) specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

\(\text{Category}\) is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\(\text{Include}\) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{stat.RegEqn})</td>
<td>Median-median line equation: (m \cdot x+b)</td>
</tr>
<tr>
<td>(\text{stat.m, stat.b})</td>
<td>Model coefficients</td>
</tr>
<tr>
<td>(\text{stat.Resid})</td>
<td>Residuals from the median-median line</td>
</tr>
<tr>
<td>(\text{stat.XReg})</td>
<td>List of data points in the modified (X) List actually used in the regression based on restrictions of (Freq,\ Category) List, and (\text{Include}) Categories</td>
</tr>
<tr>
<td>(\text{stat.YReg})</td>
<td>List of data points in the modified (Y) List actually used in the regression based on restrictions of (Freq,\ Category) List, and (\text{Include}) Categories</td>
</tr>
<tr>
<td>(\text{stat.FreqReg})</td>
<td>List of frequencies corresponding to (\text{stat.XReg}) and (\text{stat.YReg})</td>
</tr>
</tbody>
</table>

### mid()

**mid(\(\text{sourceString}, \text{Start}, \text{Count}\)) \Rightarrow \text{string}**

Returns \(\text{Count}\) characters from character string \(\text{sourceString}\), beginning with character number \(\text{Start}\).

If \(\text{Count}\) is omitted or is greater than the dimension of \(\text{sourceString}\), returns all characters from \(\text{sourceString}\), beginning with character number \(\text{Start}\).

\(\text{Count}\) must be \(\geq 0\). If \(\text{Count} = 0\), returns an empty string.

mid(“Hello there”,2) \(\Rightarrow\) "ello there"
mid(“Hello there”,7,3) \(\Rightarrow\) "the"
mid(“Hello there”,1,5) \(\Rightarrow\) "Hello"
mid(“Hello there”,1,0) \(\Rightarrow\) "Hello"
\[
\text{mid}() \quad \text{Catalog > }
\]

- \text{mid(sourceList, Start \{, Count\} \Rightarrow list}

Returns \text{Count} elements from sourceList, beginning with element number \text{Start}.

If \text{Count} is omitted or is greater than the dimension of sourceList, returns all elements from sourceList, beginning with element number \text{Start}.

\text{Count} must be \geq 0. If \text{Count} = 0, returns an empty list.

- \text{mid(sourceStringList, Start\{, Count\} \Rightarrow list}

Returns \text{Count} strings from the list of strings sourceStringList, beginning with element number \text{Start}.

\[
\begin{align*}
\text{mid}\{9,8,7,6\},3 \Rightarrow \{7,6\} \\
\text{mid}\{9,8,7,6\},2,2 \Rightarrow \{8,7\} \\
\text{mid}\{9,8,7,6\},1,2 \Rightarrow \{9,8\} \\
\text{mid}\{9,8,7,6\},1,0 \Rightarrow \{\} \\
\text{mid}\{"A","B","C","D\},2,2 \Rightarrow \{"B","C"\}
\end{align*}
\]

\[
\begin{align*}
\text{min()} \quad \text{Catalog > }
\]

- \text{min(Value1, Value2) \Rightarrow expression}

- \text{min(List1, List2) \Rightarrow list}

- \text{min(Matrix1, Matrix2) \Rightarrow matrix}

Returns the minimum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the minimum value of each pair of corresponding elements.

- \text{min(List) \Rightarrow expression}

Returns the minimum element of List.

- \text{min(Matrix) \Rightarrow matrix}

Returns a row vector containing the minimum element of each column in Matrix.

**Note:** See also \text{max()}.

\[
\begin{align*}
\text{min}\{2,3,1,4\} \Rightarrow 1.4 \\
\text{min}\{1,2\},\{-4,3\} \Rightarrow \{-4,2\} \\
\text{min}\{0,1,-7,1,3,0.5\} \Rightarrow -7 \\
\text{min}\{\begin{pmatrix} 1 & \cdot & 3 \\ -4 & 0 & 7 \end{pmatrix}\} \Rightarrow \begin{pmatrix} -4 & \cdot & 3 \\ 0 & 3 \end{pmatrix}
\end{align*}
\]

\[
\text{mirr()} \quad \text{Catalog > }
\]

- \text{mirr(financeRate, reinvestRate, CF0, CFList\{, CFFreq\})}

Financial function that returns the modified internal rate of return of an investment.

\text{financeRate} is the interest rate that you pay on the cash flow amounts.

\text{reinvestRate} is the interest rate at which the cash flows are reinvested.

\text{CF0} is the initial cash flow at time 0; it must be a real number.

\text{CFList} is a list of cash flow amounts after the initial cash flow \text{CF0}.

\text{CFFreq} is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of \text{CFList}. The default is 1; if you enter values, they must be positive integers < 10,000.

**Note:** See also \text{irr()}, page 49.

\[
\begin{align*}
\text{list1:=}\{6000,-8000,2000,-3000\} \\
\text{list2:=}\{2,2,2,1\} \\
\text{mirr}\{4.65,12,5000,\text{list1, list2}\} \Rightarrow 13.41608607
\end{align*}
\]
**mod()**

mod(Value1, Value2) ⇒ expression
mod(List1, List2) ⇒ list
mod(Matrix1, Matrix2) ⇒ matrix

Returns the first argument modulo the second argument as defined by the identities:

\[
\text{mod}(x,0) = x \\
\text{mod}(x,y) = x - y \lfloor x/y \rfloor
\]

When the second argument is non-zero, the result is periodic in that argument. The result is either zero or has the same sign as the second argument.

If the arguments are two lists or two matrices, returns a list or matrix containing the modulo of each pair of corresponding elements.

**Note:** See also remain(), page 84

**mRow()**

mRow(Value, Matrix1, Index) ⇒ matrix

Returns a copy of Matrix1 with each element in row Index of Matrix1 multiplied by Value.

**mRowAdd()**

mRowAdd(Value, Matrix1, Index1, Index2) ⇒ matrix

Returns a copy of Matrix1 with each element in row Index2 of Matrix1 replaced with:

\[
\text{Value} \cdot \text{row}_{\text{Index1}} + \text{row}_{\text{Index2}}
\]

**MultReg**

MultReg \([Y, X1, X2, X3, \ldots, X10]])

Calculates multiple linear regression of list \(Y\) on lists \(X1, X2, \ldots, X10\).

A summary of results is stored in the stat.results variable. (See page 98.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: (b0+b1 \cdot x1+b2 \cdot x2+ \ldots)</td>
</tr>
<tr>
<td>stat.b0, stat.b1, ...</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.(\tilde{y})List</td>
<td>(\tilde{y}) List = (b0+b1 \cdot x1+ \ldots)</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>
MultRegIntervals

MultRegIntervals \( Y, X1, X2, X3, \ldots, X10, [XValList], [CLevel] \)

Computes a predicted y-value, a level C prediction interval for a single observation, and a level C confidence interval for the mean response.

A summary of results is stored in the `stat.results` variable. (See page 98.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.RegEqn</code></td>
<td>Regression Equation: ( b0+ b1 \cdot x1 + b2 \cdot x2 + \ldots )</td>
</tr>
<tr>
<td><code>stat.y</code></td>
<td>A point estimate: ( \hat{y} = b0 + b1 \cdot x1 + \ldots ) for ( XValList )</td>
</tr>
<tr>
<td><code>stat.dError</code></td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td><code>stat.CLower, stat.CUpper</code></td>
<td>Confidence interval for a mean response</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td><code>stat.SE</code></td>
<td>Standard error of mean response</td>
</tr>
<tr>
<td><code>stat.LowerPred</code>, <code>stat.UpperPred</code></td>
<td>Prediction interval for a single observation</td>
</tr>
<tr>
<td><code>stat.MEPred</code></td>
<td>Prediction interval margin of error</td>
</tr>
<tr>
<td><code>stat.SEPred</code></td>
<td>Standard error for prediction</td>
</tr>
<tr>
<td><code>stat.bList</code></td>
<td>List of regression coefficients, ( {b0, b1, b2, \ldots} )</td>
</tr>
<tr>
<td><code>stat.Resid</code></td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

MultRegTests

MultRegTests \( Y, X1, X2, X3, \ldots, X10, [XValList] \)

Multiple linear regression test computes a multiple linear regression on the given data and provides the global \( F \) test statistic and \( t \) test statistics for the coefficients.

A summary of results is stored in the `stat.results` variable. (See page 98.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.RegEqn</code></td>
<td>Regression Equation: ( b0+ b1 \cdot x1 + b2 \cdot x2 + \ldots )</td>
</tr>
<tr>
<td><code>stat.F</code></td>
<td>Global ( F ) test statistic</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>P-value associated with global ( F ) statistic</td>
</tr>
<tr>
<td><code>stat.R^2</code></td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.AdjR²</td>
<td>Adjusted coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
<tr>
<td>stat.DW</td>
<td>Durbin-Watson statistic; used to determine whether first-order auto correlation is present in the model</td>
</tr>
<tr>
<td>stat.dfReg</td>
<td>Regression degrees of freedom</td>
</tr>
<tr>
<td>stat.SSReg</td>
<td>Regression sum of squares</td>
</tr>
<tr>
<td>stat.MSReg</td>
<td>Regression mean square</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td>stat.SSError</td>
<td>Error sum of squares</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Error mean square</td>
</tr>
<tr>
<td>stat.bList</td>
<td>{b0,b1,...} List of coefficients</td>
</tr>
<tr>
<td>stat.tList</td>
<td>List of t statistics, one for each coefficient in the bList</td>
</tr>
<tr>
<td>stat.PList</td>
<td>List P-values for each t statistic</td>
</tr>
<tr>
<td>stat.SEList</td>
<td>List of standard errors for coefficients in bList</td>
</tr>
<tr>
<td>stat.yList</td>
<td>( y \text{List} = b_0 + b_1 \cdot x_1 + \ldots )</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.sResid</td>
<td>Standardized residuals; obtained by dividing a residual by its standard deviation</td>
</tr>
<tr>
<td>stat.CookDist</td>
<td>Cook's distance; measure of the influence of an observation based on the residual and leverage</td>
</tr>
<tr>
<td>stat.Leverage</td>
<td>Measure of how far the values of the independent variable are from their mean values</td>
</tr>
</tbody>
</table>

### nand

| BooleanExpr1 nand BooleanExpr2 | returns Boolean expression |
| BooleanList1 nand BooleanList2 | returns Boolean list |
| BooleanMatrix1 nand BooleanMatrix2 | returns Boolean matrix |

Returns the negation of a logical and operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Integer1 nand Integer2 \( \Rightarrow \) integer**

Compares two real integers bit-by-bit using a **nand** operation.

Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).
nCr() \n
\[ nCr(\text{Value}_1, \text{Value}_2) \Rightarrow \text{expression} \]

For integer \text{Value}_1 and \text{Value}_2 with \text{Value}_1 \geq \text{Value}_2 \geq 0, \text{nCr()} is the number of combinations of \text{Value}_1 things taken \text{Value}_2 at a time. (This is also known as a binomial coefficient.)

\[ nCr(\text{Value}, 0) \Rightarrow 1 \]

\[ nCr(\text{Value}, \text{negInteger}) \Rightarrow 0 \]

\[ nCr(\text{Value}, \text{posInteger}) \Rightarrow \text{Value} \cdot \text{Value}^{-1} \cdot \ldots \cdot \text{Value}^{-\text{posInteger}+1}/\text{posInteger}! \]

\[ nCr(\text{Value}, \text{nonInteger}) \Rightarrow \text{expression}/(\text{Value}^{-\text{nonInteger}}\cdot \text{nonInteger}!) \]

\[ nCr(\text{List}_1, \text{List}_2) \Rightarrow \text{list} \]

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

\[ nCr(\text{Matrix}_1, \text{Matrix}_2) \Rightarrow \text{matrix} \]

Returns a matrix of combinations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

nDerivative() \n
\[ \text{nDerivative}(\text{Expr}_1, \text{Var} = \text{Value}_{\text{Order}}) \Rightarrow \text{value} \]

\[ \text{nDerivative}(\text{Expr}_1, \text{Var}, \text{Order}) | \text{Var} = \text{Value} \Rightarrow \text{value} \]

Returns the numerical derivative calculated using auto differentiation methods.

When \text{Value} is specified, it overrides any prior variable assignment or any current "|" substitution for the variable.

If the variable \text{Var} does not contain a numeric value, you must provide \text{Value}.

Order of the derivative must be 1 or 2.

**Note:** The \text{nDerivative()} algorithm has a limitation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of \(x \cdot (x^2+x)^{1/3}\) at \(x=0\) is equal to 0. However, because the first derivative of the subexpression \((x^2+x)^{1/3}\) is undefined at \(x=0\), and this value is used to calculate the derivative of the total expression, \text{nDerivative()} reports the result as undefined and displays a warning message.

If you encounter this limitation, verify the solution graphically. You can also try using \text{centralDiff()}.
**newMat()**

\( \text{newMat}(\text{numRows}, \text{numColumns}) \Rightarrow \text{matrix} \)

Returns a matrix of zeros with the dimension numRows by numColumns.

\[
\text{newMat}(2,3) = \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
\]

**nfMax()**

\( \text{nfMax}(\text{Expr}, \text{Var}) \Rightarrow \text{value} \)

\( \text{nfMax}(\text{Expr}, \text{Var}, \text{lowBound}) \Rightarrow \text{value} \)

\( \text{nfMax}(\text{Expr}, \text{Var}, \text{lowBound}, \text{upBound}) \Rightarrow \text{value} \)

\( \text{nfMax}(\text{Expr}, \text{Var}) | \text{lowBound} \leq \text{Var} \leq \text{upBound} \Rightarrow \text{value} \)

Returns a candidate numerical value of variable \( \text{Var} \) where the local maximum of \( \text{Expr} \) occurs.

If you supply \( \text{lowBound} \) and \( \text{upBound} \), the function looks in the closed interval \([\text{lowBound}, \text{upBound}]\) for the local maximum.

\[
\text{nfMax}(x^2 - 2x - 1, x) \Rightarrow -1. \\
\text{nfMax}(0.5x^3 - x - 2, x, 5, 5) \Rightarrow -0.816497
\]

**nfMin()**

\( \text{nfMin}(\text{Expr}, \text{Var}) \Rightarrow \text{value} \)

\( \text{nfMin}(\text{Expr}, \text{Var}, \text{lowBound}) \Rightarrow \text{value} \)

\( \text{nfMin}(\text{Expr}, \text{Var}, \text{lowBound}, \text{upBound}) \Rightarrow \text{value} \)

\( \text{nfMin}(\text{Expr}, \text{Var}) | \text{lowBound} \leq \text{Var} \leq \text{upBound} \Rightarrow \text{value} \)

Returns a candidate numerical value of variable \( \text{Var} \) where the local minimum of \( \text{Expr} \) occurs.

If you supply \( \text{lowBound} \) and \( \text{upBound} \), the function looks in the closed interval \([\text{lowBound}, \text{upBound}]\) for the local minimum.

\[
\text{nfMin}(x^2 + 2x + 5, x) \Rightarrow -1. \\
\text{nfMin}(0.5x^3 - x - 2, x, 5, 5) \Rightarrow 0.816497
\]

**nInt()**

\( \text{nInt}(\text{Expr1}, \text{Var}, \text{Lower}, \text{Upper}) \Rightarrow \text{expression} \)

If the integrand \( \text{Expr1} \) contains no variable other than \( \text{Var} \), and if \( \text{Lower} \) and \( \text{Upper} \) are constants, positive \( \infty \), or negative \( \infty \), then \( \text{nInt()} \) returns an approximation of \( \int_{\text{Lower}}^{\text{Upper}} \text{Expr1} \, \text{dVar} \). This approximation is a weighted average of some sample values of the integrand in the interval \( \text{Lower} < \text{Var} < \text{Upper} \).

The goal is six significant digits. The adaptive algorithm terminates when it seems likely that the goal has been achieved, or when it seems unlikely that additional samples will yield a worthwhile improvement.

A warning is displayed ("Questionable accuracy") when it seems that the goal has not been achieved.

Nest \( \text{nInt()} \) to do multiple numeric integration. Integration limits can depend on integration variables outside them.

\[
\text{nInt}(e^{-x^2}, x, 1, 1) \Rightarrow 1.49365 \\
\text{nInt}(\cos(x), x, \pi, \pi + 1.\times \varepsilon, 12) \Rightarrow -1.04144 \times 12 \\
\text{nInt}(\text{nInt}(e^{-x^2}, x, 0, 1), x, 0, 1) \Rightarrow 3.0423
\]
nom()  

Financial function that converts the annual effective interest rate effectiveRate to a nominal rate, given CpY as the number of compounding periods per year.

effectiveRate must be a real number, and CpY must be a real number > 0.

Note: See also eff(), page 32.

nor()  

 Returns the negation of a logical or operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Integer1 nor Integer2 ⇒ integer

Compares two real integers bit-by-bit using a nor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

norm()  

Returns the Frobenius norm.

norm(Matrix) ⇒ expression

norm(Vector) ⇒ expression

Computes the normal distribution probability between lowBound and upBound for the specified μ (default=0) and σ (default=1).

For P(X ≤ upBound), set lowBound = -9E999.

Computes the probability density function for the normal distribution at a specified XVal value for the specified μ and σ.
not

**not BooleanExpr → Boolean expression**

Returns true, false, or a simplified form of the argument.

not \( \text{Integer1} \) → integer

Returns the one’s complement of a real integer. Internally, \( \text{Integer1} \) is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one’s complement. Results are displayed according to the Base mode.

You can enter the integer in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, the integer is treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see **Base2**, page 12.

\[ 
\begin{array}{|c|}
\hline
\text{Boolean expression} & \text{result} \\
\hline
\text{not \( 2 \geq 3 \)} & \text{true} \\
\text{not \( 0b10\)} & \text{Base16} \\
\text{not \( 0hFFFFFFFFFFFFF4F \)} & 2 \\
\text{not \( 2 \)} & \text{0} \\
\hline
\end{array}
\]

In Hex base mode:

**Important:** Zero, not the letter O.

\[ 
\begin{array}{|c|}
\hline
\text{Integer1} & \text{Base16} \\
\hline
\text{not \( 0h7AC36 \)} & \text{0hFFFFFF853C9} \\
\hline
\end{array}
\]

In Bin base mode:

\[ 
\begin{array}{|c|}
\hline
\text{Value1} & \text{Base10} \\
\hline
\text{not \( 0b100101 \)} & 37 \\
\hline
\text{not \( 0b100101 \)} & \text{0b11111111111111111111111111111111} \\
\hline
\text{not \( 0b100101 \)} & \text{Base10} \\
\hline
\text{not \( 0b100101 \)} & -38 \\
\hline
\end{array}
\]

To see the entire result, press \( \blacktriangleright \) and then use \( \blacktriangleleft \) and \( \triangleright \) to move the cursor.

**Note:** A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

\[ 
\begin{array}{|c|}
\hline
nPr() & \text{Catalog >} \\
\hline
nPr(\text{Value1, Value2}) → \text{expression} \\
\hline
nPr(\text{Value, 0}) → 1 \\
nPr(\text{Value, negInteger}) → \frac{1}{((\text{Value}+1) \cdot (\text{Value}+2)\ldots (\text{Value}-\text{negInteger}))} \\
nPr(\text{Value, posInteger}) → \text{Value} \cdot (\text{Value}+1)\ldots (\text{Value}+\text{posInteger}+1) \\
nPr(\text{Value, nonInteger}) → \text{Value}! / (\text{Value}-\text{nonInteger})! \\
nPr(\text{List1, List2}) → \text{list} \\
nPr(\text{Matrix1, Matrix2}) → \text{matrix} \\
\hline
\end{array}
\]

\[ 
\begin{array}{|c|}
\hline
\text{Value1} & \text{Value2} & \text{Result} \\
\hline
z & 5 & 60 \\
(z, 3) & z & 120 \\
\{5, 4, 3\}, \{2, 4, 2\} & 20, 24, 6 \\
\{6, 5\}, \{2, 2\} & 30, 20 \\
\{4, 3\}, \{2, 2\} & 12, 6 \\
\{5, 4, 3\}, \{2, 4, 2\} & 20, 24, 6 \\
\{6, 5\}, \{2, 2\} & 30, 20 \\
\{4, 3\}, \{2, 2\} & 12, 6 \\
\hline
\end{array}
\]
**npv()**

Financial function that calculates net present value; the sum of the present values for the cash inflows and outflows. A positive result for npv indicates a profitable investment.

*InterestRate* is the rate by which to discount the cash flows (the cost of money) over one period.

*CF0* is the initial cash flow at time 0; it must be a real number.

*CFList* is a list of cash flow amounts after the initial cash flow *CF0*.

*CFFreq* is a list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.

**nSolve()**

Iteratively searches for one approximate real numeric solution to *Equation* for its one variable. Specify the variable as:

- variable
- variable = real number

For example, x is valid and so is x=3.

nSolve() attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string "no solution found."

**npv()**

`npv(InterestRate, CF0, CFList, CFFreq)`

- **list1:** `{6000, 8000, 2000, -3000}`
- **list2:** `{2, 2, 2, 1}`

`npv(10, 5000, list1, list2)`

4769.91

**nSolve()**

- `nSolve(x^2 + 5 \cdot x - 25 = 9, x)`
  - `3.84429`
- `nSolve(x^2 = 4, x = 1)`
  - `-2`
- `nSolve(x^2 = 4, x = 1)`
  - `2`

**Note:** If there are multiple solutions, you can use a guess to help find a particular solution.

- `nSolve(x^2 + 5 \cdot x - 25 = 9, x)`
  - `x < 0`
  - `-8.84429`
- `nSolve((1 + r)^{24} - 26, r)`
  - `r > 0` and `r < 0.25`
  - `0.006886`
- `nSolve(x^2 = -1, x)`
  - "No solution found"
OneVar

OneVar \{1\}, \{X1\}, \{Freq\}, \{Category\}, \{Include\}]
OneVar \{n\}, \{X1\}, \{X2\}, \{X3\}, \ldots, \{X20\}]

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the \textit{stat.results} variable. (See page 98.)

All the lists must have equal dimension except for \textit{Include}.

\textit{Freq} is an optional list of frequency values. Each element in \textit{Freq} specifies the frequency of occurrence for each corresponding \textit{X} and \textit{Y} data point. The default value is 1. All elements must be integers \geq 0.

\textit{Category} is a list of numeric category codes for the corresponding \textit{X} values.

\textit{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists \textit{X}, \textit{Freq}, or \textit{Category} results in a void for the corresponding element of all those lists. An empty element in any of the lists \textit{X1} through \textit{X20} results in a void for the corresponding element of all those lists. For more information on empty elements, see page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{stat.X}</td>
<td>Mean of \textit{x} values</td>
</tr>
<tr>
<td>\textit{stat.Σx}</td>
<td>Sum of \textit{x} values</td>
</tr>
<tr>
<td>\textit{stat.Σx^2}</td>
<td>Sum of \textit{x^2} values</td>
</tr>
<tr>
<td>\textit{stat.ss}</td>
<td>Sample standard deviation of \textit{x}</td>
</tr>
<tr>
<td>\textit{stat.sx}</td>
<td>Population standard deviation of \textit{x}</td>
</tr>
<tr>
<td>\textit{stat.n}</td>
<td>Number of data points</td>
</tr>
<tr>
<td>\textit{stat.MinX}</td>
<td>Minimum of \textit{x} values</td>
</tr>
<tr>
<td>\textit{stat.Q_1}</td>
<td>1st Quartile of \textit{x}</td>
</tr>
<tr>
<td>\textit{stat.MedianX}</td>
<td>Median of \textit{x}</td>
</tr>
<tr>
<td>\textit{stat.Q_2}</td>
<td>3rd Quartile of \textit{x}</td>
</tr>
<tr>
<td>\textit{stat.MaxX}</td>
<td>Maximum of \textit{x} values</td>
</tr>
<tr>
<td>\textit{stat.SSX}</td>
<td>Sum of squares of deviations from the mean of \textit{x}</td>
</tr>
</tbody>
</table>
or

BooleanExpr1 or BooleanExpr2 returns Boolean expression
BooleanList1 or BooleanList2 returns Boolean list
BooleanMatrix1 or BooleanMatrix2 returns Boolean matrix

Returns true or false or a simplified form of the original entry.

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See xor.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \[ \text{→} \] instead of \[ \text{enter} \] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Integer1 or Integer2 \[ \Rightarrow \] integer

Compares two real integers bit-by-bit using an or operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see PBase2, page 12.

Note: See xor.

ord()

ord(String) \[ \Rightarrow \] integer
ord(List1) \[ \Rightarrow \] list

Returns the numeric code of the first character in character string String, or a list of the first characters of each list element.

PPRx()

PPRx(rExpr, \[ \theta \]Expr) \[ \Rightarrow \] expression
PPRx(rList, \[ \theta \]List) \[ \Rightarrow \] list
PPRx(rMatrix, \[ \theta \]Matrix) \[ \Rightarrow \] matrix

Returns the equivalent x-coordinate of the \((r, \theta)\) pair.

Note: The \(\theta\) argument is interpreted as either a degree, gradian or radian angle, according to the current angle mode. If the argument is an expression, you can use \(^{\circ}\), \(\text{g}\) or \(\text{r}\) to override the angle mode setting temporarily.

Note: You can insert this function from the computer keyboard by typing \(\text{P} \theta \text{r} \text{p} \text{x} (...) \).

In Radian angle mode:

\[
\text{PPRx}\left\{\frac{\pi}{3}, \frac{\pi}{4}, 0 \right\} = \{-1.5, 0.7107, 1.3\}
\]
**P Ry()**

Returns the equivalent y-coordinate of the \((r, \theta)\) pair.

**Note:** The \(\theta\) argument is interpreted as either a degree, radian or gradian angle, according to the current angle mode.

**Note:** You can insert this function from the computer keyboard by typing `P@>Ry( ... )`.

---

**PassErr**

Passes an error to the next level.

If system variable `errCode` is zero, **PassErr** does not do anything.

The **Else** clause of the **Try...Else...EndTry** block should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. If there are no more pending **Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.

**Note:** See also **ClrErr**, page 17, and **Try**, page 106.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `·` at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

For an example of **PassErr**, see Example 2 under the **Try** command, page 106.

---

**piecewise()**

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

**Note:** See also **Piecewise template**, page 2.

---

**poissCdf()**

Computes a cumulative probability for the discrete Poisson distribution with specified mean \(\lambda\).

For \(P(X \leq \text{upBound})\), set \(\text{lowBound}=0\)

---

**poissPdf()**

Computes a probability for the discrete Poisson distribution with the specified mean \(\lambda\).
\textbf{\textit{Polar}}

\textbf{\textit{Vector \(bindingkey{Polar})}}

\textbf{Note:} You can insert this operator from the computer keyboard by typing \texttt{\textasciitilde \textbf{Polar}}.

Displays \textit{vector} in polar form \([r \angle \theta]\). The vector must be of dimension 2 and can be a row or a column.

\textbf{Note:} \texttt{\textbf{Polar}} is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update \textit{ans}.

\textbf{Note:} See also \texttt{\textbf{Rect}}, page 82.

\textit{complexValue \boundingkey{Polar}}

Displays \textit{complexVector} in polar form.

- Degree angle mode returns \([r \angle \theta]\).
- Radian angle mode returns \(re^{\theta i}\).

\textit{complexValue} can have any complex form. However, an \(re^{\theta i}\) entry causes an error in Degree angle mode.

\textbf{Note:} You must use the parentheses for an \([r \angle \theta]\) polar entry.

\begin{align*}
\text{In Radian angle mode:} & \quad (3+4\cdot i)\text{\textasciitilde \textbf{Polar}} \quad e^{92795\cdot i - 5} \\
(4 \angle \frac{\pi}{3})\text{\textasciitilde \textbf{Polar}} \quad e^{1.0472\cdot i - 4}.
\end{align*}

\begin{align*}
\text{In Gradian angle mode:} & \quad (4\cdot i)\text{\textasciitilde \textbf{Polar}} \quad (4 \angle 100) \\
\text{In Degree angle mode:} & \quad (3+4\cdot i)\text{\textasciitilde \textbf{Polar}} \quad (5 \angle 53.1301)
\end{align*}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\texttt{\textbf{polyEval(}} & \texttt{\textbf{polyEval(List1, Expr1) \Rightarrow expression}} \\
\hline
\texttt{\textbf{polyEval(List1, List2) \Rightarrow expression}} & \texttt{polyEval(\{1,2,3,4\},2)} \quad 26 \\
\texttt{polyEval(\{1,2,3,4\},\{2,7\})} & \{26, 262\} \\
\hline
\end{tabular}
\caption{\texttt{\textbf{polyEval()}}}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\texttt{\textbf{polyRoots(}} & \texttt{\textbf{polyRoots(Poly, Var) \Rightarrow list}} \\
\hline
\texttt{\textbf{polyRoots(ListOfCoeffs) \Rightarrow list}} & \texttt{polyRoots(\texttt{y}^3+1, \texttt{y})} \quad \{-1\} \\
\texttt{\textbf{polyRoots(ListOfCoeffs)}} & \texttt{cPolyRoots(\texttt{y}^3+1, \texttt{y})} \quad \{-1, 0.5-0.8660254\cdot i, 0.5+0.8660254\cdot i\} \\
\texttt{\textbf{polyRoots(x^2+2\cdot x+1, x)}} & \texttt{polyRoots(\texttt{1,2,1})} \quad \{-1, -1\} \\
\texttt{\textbf{polyRoots(\texttt{1,2,1})}} & \texttt{\{-1, -1\}} \\
\hline
\end{tabular}
\caption{\texttt{\textbf{polyRoots()}}}
\end{table}

\texttt{\textbf{polyEval(}} List1, \texttt{Expr1) \Rightarrow expression}

\texttt{\textbf{polyEval(List1, List2) \Rightarrow expression}}

Interprets the first argument as the coefficient of a descending-degree polynomial, and returns the polynomial evaluated for the value of the second argument.

\texttt{\textbf{polyRoots(Poly, Var) \Rightarrow list}}

\texttt{\textbf{polyRoots(ListOfCoeffs) \Rightarrow list}}

The first syntax, \texttt{\textbf{polyRoots(Poly, Var)}}, returns a list of real roots of polynomial \textit{Poly} with respect to variable \textit{Var}. If no real roots exist, returns an empty list: \{\}. \textit{Poly} must be a polynomial in expanded form in one variable. Do not use unexpanded forms such as \(y^2 + y + 1\) or \(x^2 + x + 1\).

The second syntax, \texttt{\textbf{polyRoots(ListOfCoeffs)}}, returns a list of real roots for the coefficients in \textit{ListOfCoeffs}.

\textbf{Note:} See also \texttt{\textbf{cPolyRoots()}}, page 23.
**PowerReg**

**Catalog > PowerReg**

`PowerReg X, Y [, Freq] [, Category, Include]]`

Computes the power regression $y = (a \cdot (x)^b)$ on lists $X$ and $Y$ with frequency $Freq$. A summary of results is stored in the `stat.results` variable. (See page 98.)

All the lists must have equal dimension except for `Include`.

$X$ and $Y$ are lists of independent and dependent variables.

`Freq` is an optional list of frequency values. Each element in `Freq` specifies the frequency of occurrence for each corresponding $X$ and $Y$ data point. The default value is 1. All elements must be integers $\geq 0$.

`Category` is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.

`Include` is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.ReqEqn</code></td>
<td>Regression equation: $a \cdot (x)^b$</td>
</tr>
<tr>
<td><code>stat.a, stat.b</code></td>
<td>Regression coefficients</td>
</tr>
<tr>
<td><code>stat.r^2</code></td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td><code>stat.r</code></td>
<td>Correlation coefficient for transformed data ($\ln(x), \ln(y)$)</td>
</tr>
<tr>
<td><code>stat.Resid</code></td>
<td>Residuals associated with the power model</td>
</tr>
<tr>
<td><code>stat.ResidTrans</code></td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td><code>stat.XReg</code></td>
<td>List of data points in the modified $X$ List actually used in the regression based on restrictions of $Freq$, <code>Category List</code>, and <code>Include Categories</code></td>
</tr>
<tr>
<td><code>stat.YReg</code></td>
<td>List of data points in the modified $Y$ List actually used in the regression based on restrictions of $Freq$, <code>Category List</code>, and <code>Include Categories</code></td>
</tr>
<tr>
<td><code>stat.FreqReg</code></td>
<td>List of frequencies corresponding to <code>stat.XReg</code> and <code>stat.YReg</code></td>
</tr>
</tbody>
</table>
**Prgm**

**Block**

**EndPrgm**

Template for creating a user-defined program. Must be used with the Define, Define LibPub, or Define LibPriv command.

*Block* can be a single statement, a series of statements separated with the ";" character, or a series of statements on separate lines.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \[\text{enter}\] instead of \[\text{\textbf{\text{k}} enter}\] at the end of each line. On the computer keyboard, hold down \[\text{Alt} \text{ and press Enter.}\]

---

**Calculate GCD and display intermediate results.**

Define `proggcd(a, b) = Prgm`

```plaintext```
Local d
While b ≠ 0
  d := mod(a, b)
  a := b
  b := d
  Disp a, " ", b
EndWhile
Disp "GCD = ", a
EndPrgm
```

---

`proggcd(4560, 450)`

```
450 60
60 30
30 0
GCD = 30
```

---

**prodSeq()**

See \[\prod\](), page 125.

---

**Product (\(\Pi\))**

See \[\prod\](), page 125.

---

**product()**

**product(List[, Start[, End]]) \(\Rightarrow\) expression**

Returns the product of the elements contained in List. Start and End are optional. They specify a range of elements.

**product(Matrix[, Start[, End]]) \(\Rightarrow\) matrix**

Returns a row vector containing the products of the elements in the columns of Matrix. Start and End are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 132.
propFrac()\[\text{Catalog > propFrac}\]

\begin{align*}
\text{propFrac} & (\text{Value1}, \text{Var}) \Rightarrow \text{value} \\
\text{propFrac} & (\text{rational\_number}) \text{ returns rational\_number as the sum of an integer and a fraction having the same sign and a greater denominator magnitude than numerator magnitude.}
\end{align*}

\begin{align*}
\text{propFrac} & \left(\frac{4}{3}\right) \quad \Rightarrow \quad 1 + \frac{1}{3} \\
\text{propFrac} & \left(-\frac{4}{3}\right) \quad \Rightarrow \quad -1 - \frac{1}{3}
\end{align*}

\text{propFrac}(\text{rational\_expression}, \text{Var}) \text{ returns the sum of proper ratios and a polynomial with respect to Var. The degree of Var in the denominator exceeds the degree of Var in the numerator in each proper ratio. Similar powers of Var are collected. The terms and their factors are sorted with Var as the main variable.}

If Var is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.

You can use the \text{propFrac()} function to represent mixed fractions and demonstrate addition and subtraction of mixed fractions.

\begin{align*}
\text{propFrac} & \left(\frac{11}{7}\right) \quad \Rightarrow \quad 1 + \frac{4}{7} \\
\text{propFrac} & \left(\frac{3 + \frac{1}{11} + 5 + \frac{3}{4}}{11}\right) \quad \Rightarrow \quad 8 + \frac{37}{44} \\
\text{propFrac} & \left(\frac{3 + \frac{1}{11} + 5 + \frac{3}{4}}{11}\right) \quad \Rightarrow \quad -2 - \frac{29}{44}
\end{align*}

\textbf{QR}

\begin{align*}
\text{QR Matrix, qMatrix, rMatrix, Tol} \\
\text{Catalog > QR}
\end{align*}

\text{QR} \text{ calculates the Householder QR factorization of a real or complex matrix. The resulting Q and R matrices are stored to the specified Matrix. The Q matrix is unitary. The R matrix is upper triangular.}

Optionally, any matrix element is treated as zero if its absolute value is less than \text{Tol}. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, \text{Tol} is ignored.

- If you use \text{ctrl enter} or set the \text{Auto or Approximate} mode to Approximate, computations are done using floating-point arithmetic.
- If \text{Tol} is omitted or not used, the default tolerance is calculated as:
  \[5\times 10^{-14} \times \max(\text{dim}(\text{Matrix})) \times \text{rowNorm}(\text{Matrix})\]

\begin{align*}
\text{QR m1, qm, rm} \quad \text{Done}
\end{align*}

\begin{align*}
\text{qm} & \begin{bmatrix} 0.123091 & 0.904534 & 0.408248 \\ 0.492366 & 0.301511 & 0.816497 \\ 0.86164 & -0.301511 & 0.408248 \end{bmatrix} \\
\text{rm} & \begin{bmatrix} 8.12404 & 9.60114 & 11.0782 \\ 0 & 0.904534 & 1.80907 \\ 0 & 0 & 0 \end{bmatrix}
\end{align*}

\text{ClearAZ. \quad Done}

\text{The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in qMaxName are the orthonormal basis vectors that span the space defined by matrix.}
### QuadReg

**Catalog >**

#### QuadReg 

Computes the quadratic polynomial regression \( y = a \cdot x^2 + b \cdot x + c \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the `stat.results` variable. (See page 98.)

All the lists must have equal dimension except for `Include`.

\( X \) and \( Y \) are lists of independent and dependent variables.

`Freq` is an optional list of frequency values. Each element in `Freq` specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

`Category` is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

`Include` is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.ReqEqn</code></td>
<td>Regression equation: ( a \cdot x^2 + b \cdot x + c )</td>
</tr>
<tr>
<td><code>stat.a</code>, <code>stat.b</code>, <code>stat.c</code></td>
<td>Regression coefficients</td>
</tr>
<tr>
<td><code>stat.R^2</code></td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td><code>stat.Resid</code></td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td><code>stat.XReg</code></td>
<td>List of data points in the modified ( X ) list actually used in the regression based on restrictions of <code>Freq</code>, <code>Category List</code>, and <code>Include Categories</code></td>
</tr>
<tr>
<td><code>stat.YReg</code></td>
<td>List of data points in the modified ( Y ) list actually used in the regression based on restrictions of <code>Freq</code>, <code>Category List</code>, and <code>Include Categories</code></td>
</tr>
<tr>
<td><code>stat.FreqReg</code></td>
<td>List of frequencies corresponding to <code>stat.XReg</code> and <code>stat.YReg</code></td>
</tr>
</tbody>
</table>

### QuartReg

**Catalog >**

#### QuartReg 

Computes the quartic polynomial regression 
\[ y = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \] on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the `stat.results` variable. (See page 98.)

All the lists must have equal dimension except for `Include`.

\( X \) and \( Y \) are lists of independent and dependent variables.

`Freq` is an optional list of frequency values. Each element in `Freq` specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

`Category` is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

`Include` is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.
### Output variable | Description
--- | ---
stat.RegEqn | Regression equation: \( a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \)
stat.a, stat.b, stat.c, stat.d, stat.e | Regression coefficients
stat.\(R^2\) | Coefficient of determination
stat.Resid | Residuals from the regression
stat.XReg | List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories
stat.YReg | List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories
stat.FreqReg | List of frequencies corresponding to stat.XReg and stat.YReg

### R

#### \(R^4P\theta()\)

\[ R^4P\theta(xValue, yValue) \Rightarrow value \]
\[ R^4P\theta(xList, yList) \Rightarrow list \]
\[ R^4P\theta(xMatrix, yMatrix) \Rightarrow matrix \]

Returns the equivalent \(\theta\)-coordinate of the \((x,y)\) pair arguments.

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the computer keyboard by typing \(R^4P\theta(...).\)

#### \(R^4Pr()\)

\[ R^4Pr(xValue, yValue) \Rightarrow value \]
\[ R^4Pr(xList, yList) \Rightarrow list \]
\[ R^4Pr(xMatrix, yMatrix) \Rightarrow matrix \]

Returns the equivalent \(r\)-coordinate of the \((x,y)\) pair arguments.

**Note:** You can insert this function from the computer keyboard by typing \(R^4Pr(...).\)
**Rad**

- **Value/Rad ⇒ value**
  - Converts the argument to radian angle measure.
  - **Note:** You can insert this operator from the computer keyboard by typing `@>Rad`.

**Catalog >**

In Degree angle mode:

\[
\frac{\pi}{2} \cdot \frac{1.5}{\pi} = 0.2618
\]

In Gradian angle mode:

\[
\frac{\pi}{2} \cdot \frac{1.5}{\pi} = 0.2356
\]

---

**rand()**

- **rand() ⇒ expression**
- **rand(#Trials) ⇒ list**
  - **rand()** returns a random value between 0 and 1.
  - **rand(#Trials)** returns a list containing #Trials random values between 0 and 1.

**Catalog >**

- Sets the random-number seed.

\[
\text{RandSeed 1147} \quad \text{Done}
\]

\[
\text{rand}(2) \quad \{0.158206, 0.717917\}
\]

---

**randBin()**

- **randBin(n, p) ⇒ expression**
- **randBin(n, p, #Trials) ⇒ list**
  - **randBin(n, p)** returns a random real number from a specified Binomial distribution.
  - **randBin(n, p, #Trials)** returns a list containing #Trials random real numbers from a specified Binomial distribution.

**Catalog >**

- \[
\text{randBin}(80, 0.5) \quad 34.0
\]

- \[
\text{randBin}(80, 0.5, 3) \quad \{47.0, 41.0, 46.0\}
\]

---

**randInt()**

- **randInt(lowBound, upBound) ⇒ expression**
- **randInt(lowBound, upBound, #Trials) ⇒ list**
  - **randInt(lowBound, upBound)** returns a random integer within the range specified by lowBound and upBound integer bounds.
  - **randInt(lowBound, upBound, #Trials)** returns a list containing #Trials random integers within the specified range.

**Catalog >**

- \[
\text{randInt}(3, 10) \quad 7.0
\]

- \[
\text{randInt}(3, 10, 4) \quad \{8.0, 9.0, 4.0, 4.0\}
\]

---

**randMat()**

- **randMat(numRows, numColumns) ⇒ matrix**
  - Returns a matrix of integers between -9 and 9 of the specified dimension.
  - Both arguments must simplify to integers.

**Catalog >**

- \[
\text{RandSeed 1147} \quad \text{Done}
\]

\[
\text{randMat}(3, 3) \quad \begin{bmatrix} 8 & -3 & 6 \\ -2 & 3 & -6 \\ 0 & 4 & -6 \end{bmatrix}
\]

- **Note:** The values in this matrix will change each time you press `enter`.

---

**randNorm()**

- **randNorm(μ, σ) ⇒ expression**
- **randNorm(μ, σ, #Trials) ⇒ list**
  - **randNorm(μ, σ)** returns a decimal number from the specified normal distribution. It could be any real number but will be heavily concentrated in the interval \(\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma\).
  - **randNorm(μ, σ, #Trials)** returns a list containing #Trials decimal numbers from the specified normal distribution.

**Catalog >**

- \[
\text{RandSeed 1147} \quad \text{Done}
\]

\[
\text{randNorm}(0.1) \quad 0.492541
\]

\[
\text{randNorm}(3.45) \quad -3.54356
\]
randPoly()

randPoly(Var, Order) \Rightarrow \text{expression}

Returns a polynomial in \text{Var} of the specified \text{Order}. The coefficients are random integers in the range -9 through 9. The leading coefficient will not be zero.

\text{Order} must be 0–99.

RandSeed

RandSeed \text{Number}

If \text{Number} = 0, sets the seeds to the factory defaults for the random-number generator. If \text{Number} \neq 0, it is used to generate two seeds, which are stored in system variables seed1 and seed2.

real()

real(Value) \Rightarrow value

Returns the real part of the argument.

real(List) \Rightarrow list

Returns the real parts of all elements.

real(Matrix) \Rightarrow matrix

Returns the real parts of all elements.

Rect

\text{Vector} \Rightarrow \text{Rect}

\text{Note: You can insert this operator from the computer keyboard by typing } \Rightarrow \text{Rect}.

Displays \text{Vector} in rectangular form \([x, y, z]\). The vector must be of dimension 2 or 3 and can be a row or a column.

\text{Note: } \Rightarrow \text{Rect} \text{ is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update } \text{ans}.

\text{Note: See also } \Rightarrow \text{Polar, page 75.}
complexValue ▶ Rect

Displays complexValue in rectangular form a+bi. The complexValue can have any complex form. However, an \( \text{re}^0 \) entry causes an error in Degree angle mode.

**Note:** You must use parentheses for an \((r\angle 0)\) polar entry.

In Radian angle mode:

\[
\begin{align*}
\left\{ \frac{\pi}{3} \right\} \cdot 4 \Rightarrow \text{Rect} & \quad 11.3986 \\
\left\{ \frac{\pi}{3} \right\} \cdot 4 \Rightarrow \text{Rect} & \quad 2. + 3.4641 \cdot i
\end{align*}
\]

In Gradian angle mode:

\[
\left\{ \frac{1 \angle 100}{} \right\} \cdot \text{Rect} \Rightarrow i
\]

In Degree angle mode:

\[
\left\{ \frac{4 \angle 60}{} \right\} \cdot \text{Rect} \Rightarrow 2. + 3.4641 \cdot i
\]

**Note:** You must use parentheses for an \((r\angle 0)\) polar entry.

---

ref()

**Catalog ▶**

**ref(Matrix1, Tol) \Rightarrow matrix**

Returns the row echelon form of Matrix1.

Optionally, any matrix element is treated as zero if its absolute value is less than Tol. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, Tol is ignored.

- If you use `ct` or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as:

\[
5 \cdot 10^{-14} \cdot \max(\text{dim}(\text{Matrix1})) \cdot \text{rowNorm}(\text{Matrix1})
\]

Avoid undefined elements in Matrix1. They can lead to unexpected results.

For example, if \( a \) is undefined in the following expression, a warning message appears and the result is shown as:

\[
\begin{pmatrix}
1 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix} \Rightarrow \begin{pmatrix}
1 & \frac{1}{a} & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

The warning appears because the generalized element \(1/a\) would not be valid for \(a=0\).

You can avoid this by storing a value to \( a \) beforehand or by using the constraint ("|") operator to substitute a value, as shown in the following example.

\[
\begin{pmatrix}
1 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix} | a = 0 \Rightarrow \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{pmatrix}
\]

**Note:** See also `rref()`, page 88.
**remain()**

\[
\text{remain}(\text{Value}_1, \text{Value}_2) \Rightarrow \text{value}
\]

\[
\text{remain}([\text{List}_1], [\text{List}_2]) \Rightarrow \text{list}
\]

\[
\text{remain}([\text{Matrix}_1], [\text{Matrix}_2]) \Rightarrow \text{matrix}
\]

Returns the remainder of the first argument with respect to the second argument as defined by the identities:

\[
\text{remain}(x,0) = x
\]

\[
\text{remain}(x,y) = x - y \cdot \text{ipart}(x/y)
\]

As a consequence, note that \(\text{remain}(-x,y) = -\text{remain}(x,y)\). The result is either zero or it has the same sign as the first argument.

**Note:** See also \(\text{mod}\)(), page 64.

**Request**

\[
\text{Request} \ promptString, \text{var}[, \text{DispFlag}[, \text{statusVar}]])
\]

Programming command: Pauses the program and displays a dialog box containing the message \(\text{promptString}\) and an input box for the user’s response.

When the user types a response and clicks **OK**, the contents of the input box are assigned to variable \(\text{var}\).

If the user clicks **Cancel**, the program proceeds without accepting any input. The program uses the previous value of \(\text{var}\) if \(\text{var}\) was already defined.

The optional \(\text{DispFlag}\) argument can be any expression.

- If \(\text{DispFlag}\) is omitted or evaluates to \(1\), the prompt message and user’s response are displayed in the Calculator history.
- If \(\text{DispFlag}\) evaluates to \(0\), the prompt and response are not displayed in the history.

The optional \(\text{statusVar}\) argument gives the program a way to determine how the user dismissed the dialog box. Note that \(\text{statusVar}\) requires the \(\text{DispFlag}\) argument.

- If the user clicks **OK** or pressed Enter or Ctrl+Enter, variable \(\text{statusVar}\) is set to a value of \(1\).
- Otherwise, variable \(\text{statusVar}\) is set to a value of \(0\).

The \(\text{func}\)() argument allows a program to store the user’s response as a function definition. This syntax operates as if the user executed the command:

\[
\text{Define} \ \text{func}(\arg_1, ... \arg_n) = \text{user’s response}
\]

The program can then use the defined function \(\text{func}\)(). The \(\text{promptString}\) should guide the user to enter an appropriate user’s response that completes the function definition.

**Note:** You can use the **Request** command within a user-defined program but not within a function.

To stop a program that contains a **Request** command inside an infinite loop:

- **Windows®:** Hold down the **F12** key and press Enter repeatedly.
- **Macintosh®:** Hold down the **F5** key and press Enter repeatedly.
- **Handheld:** Hold down the **On** key and press enter repeatedly.

**Note:** See also **RequestStr**, page 85.
RequestStr

RequestStr promptString, var[, DispFlag]

Programming command: Operates identically to the first syntax of the Request command, except that the user’s response is always interpreted as a string. By contrast, the Request command interprets the response as an expression unless the user encloses it in quotation marks (" ").

Note: You can use the RequestStr command within a user-defined program but not within a function.

To stop a program that contains a RequestStr command within an infinite loop:

- Windows®: Hold down the F12 key and press Enter repeatedly.
- Macintosh®: Hold down the F5 key and press Enter repeatedly.
- handheld: Hold down the on key and press enter repeatedly.

Note: See also Request, page 84.

Define a program:

Define requestStr_demo() = Prgm
RequestStr “Your name:”, name, 0
Disp “Response has ”, dim(name), ” characters.”
EndPrgm

Run the program and type a response:

requestStr_demo()

Result after selecting OK (Note that the DispFlag argument of 0 omits the prompt and response from the history):

requestStr_demo()  Response has 5 characters.

Return

Return [Expr]

Returns Expr as the result of the function. Use within a Func...EndFunc block.

Note: Use Return without an argument within a Prgm...EndPrgm block to exit a program.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing ↵ instead of enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Define factorial (nn) = Func
Local answer, count
1 -> answer
For count, 1, nn
answer - count -> answer
EndFor
Return answer
EndFunc

Done

factorial(3)  6

right()

right(List1[, Num]) ⇒ list

Returns the rightmost Num elements contained in List1.

If you omit Num, returns all of List1.

right(sourceString[, Num]) ⇒ string

Returns the rightmost Num characters contained in character string sourceString.

If you omit Num, returns all of sourceString.

right(Comparison) ⇒ expression

Returns the right side of an equation or inequality.

right({1,3,2,4},3)  {3,2,4}

right(“Hello”,2)  “lo”
rk23()  

**Catalog >**

**rk23(Expr, Var, depVar, [Var0, VarMax], depVar0, VarStep [, difftol]) ⇒ matrix**

**rk23(SystemOfExpr, Var, ListOfDepVars, [Var0, VarMax], ListOfDepVars0, VarStep [, difftol]) ⇒ matrix**

**rk23(ListOfExpr, Var, ListOfDepVars, [Var0, VarMax], ListOfDepVars0, VarStep [, difftol]) ⇒ matrix**

Uses the Runge-Kutta method to solve the system

\[
\frac{d \text{depVar}}{d \text{Var}} = \text{Expr(Var, depVar)}
\]

with \(\text{depVar(Var0)} = \text{depVar0}\) on the interval \([\text{Var0}, \text{VarMax}]\). Returns a matrix whose first row defines the \(\text{Var}\) output values as defined by \(\text{VarStep}\). The second row defines the value of the first solution component at the corresponding \(\text{Var}\) values, and so on.

\(\text{Expr}\) is the right hand side that defines the ordinary differential equation (ODE).

\(\text{SystemOfExpr}\) is a system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in \(\text{ListOfDepVars}\)).

\(\text{ListOfExpr}\) is a list of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in \(\text{ListOfDepVars}\)).

\(\text{Var}\) is the independent variable.

\(\text{ListOfDepVars}\) is a list of dependent variables.

\([\text{Var0}, \text{VarMax}]\) is a two-element list that tells the function to integrate from \(\text{Var0}\) to \(\text{VarMax}\).

\(\text{ListOfDepVars0}\) is a list of initial values for dependent variables.

If \(\text{VarStep}\) evaluates to a nonzero number: \(\text{sign(VarStep)} = \text{sign(VarMax - Var0)}\) and solutions are returned at \(\text{Var0} + i \times \text{VarStep}\) for all \(i=0,1,2,\ldots\) such that \(\text{Var0} + i \times \text{VarStep}\) is in \([\text{Var0}, \text{VarMax}]\) (may not get a solution value at \(\text{VarMax}\)).

If \(\text{VarStep}\) evaluates to zero, solutions are returned at the “Runge-Kutta” \(\text{Var}\) values.

\(\text{diftol}\) is the error tolerance (defaults to 0.001).

**Differential equation:**

\[y’ = 0.001 \cdot y \cdot (100 - y)\] and \(y(0)=10\)

\[
\text{rk23}\left(0.001 \cdot y \cdot (100 - y), t, y, \{0,100\}, 10,1,1 \times 6\right)
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
10 & 10.9367 & 11.9493 & 13.042 & 14.2189
\end{array}
\]

To see the entire result, press \(\uparrow\) and then use \(\langle\) and \(\rangle\) to move the cursor.

Same equation with \(\text{diftol}\) set to \(1 \times E^{-6}\)

\[
\text{rk23}\left(0.001 \cdot y \cdot (100 - y), t, y, \{0,100\}, 10,1,1 \times 6\right)
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

**root()**  

**Catalog >**

**root(Value) ⇒ \text{root}**

**root(Value1, Value2) ⇒ \text{root}**

**root(Value) returns the square root of \(\text{Value}\).**

**root(Value1, Value2) returns the Value2 root of \(\text{Value1}\). \(\text{Value1}\) can be a real or complex floating point constant or an integer or complex rational constant.**

**Note:** See also \(\text{Nth root template},\) page 1.

**rotate()**  

**Catalog >**

**rotate(Integer1[,nofRotations]) ⇒ \text{integer}**

Rotates the bits in a binary integer. You can enter \(\text{Integer1}\) in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of \(\text{Integer1}\) is too large for this form, a symmetric modulo operation brings it within the range. For more information, see \(\text{Base2},\) page 12.
rotate()  

If \( \# \text{ofRotations} \) is positive, the rotation is to the left. If \( \# \text{ofRotations} \) is negative, the rotation is to the right. The default is \(-1\) (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

\[
0b00000000000001111010110000110101
\]

Rightmost bit rotates to leftmost.

produces:

\[
0b10000000000000111101011000011010
\]

The result is displayed according to the Base mode.

\[
\text{rotate}(\text{List1}[\# \text{ofRotations}]) \Rightarrow \text{list}
\]

Returns a copy of \( \text{List1} \) rotated right or left by \( \# \text{ofRotations} \) elements. Does not alter \( \text{List1} \).

If \( \# \text{ofRotations} \) is positive, the rotation is to the left. If \( \# \text{ofRotations} \) is negative, the rotation is to the right. The default is \(-1\) (rotate right one element).

\[
\text{rotate}(\text{String1}[\# \text{ofRotations}]) \Rightarrow \text{string}
\]

Returns a copy of \( \text{String1} \) rotated right or left by \( \# \text{ofRotations} \) characters. Does not alter \( \text{String1} \).

If \( \# \text{ofRotations} \) is positive, the rotation is to the left. If \( \# \text{ofRotations} \) is negative, the rotation is to the right. The default is \(-1\) (rotate right one character).

round()  

\[
\text{round}(\text{Value1}[\text{digits}]) \Rightarrow \text{value}
\]

Returns the argument rounded to the specified number of digits after the decimal point.

\( \text{digits} \) must be an integer in the range 0–12. If \( \text{digits} \) is not included, returns the argument rounded to 12 significant digits.

**Note:** Display digits mode may affect how this is displayed.

\[
\text{round}(\text{List1}[\text{digits}]) \Rightarrow \text{list}
\]

Returns a list of the elements rounded to the specified number of digits.

\[
\text{round}(\text{Matrix1}[\text{digits}]) \Rightarrow \text{matrix}
\]

Returns a matrix of the elements rounded to the specified number of digits.

rowAdd()  

\[
\text{rowAdd}(\text{Matrix1}, \text{rIndex1}, \text{rIndex2}) \Rightarrow \text{matrix}
\]

Returns a copy of \( \text{Matrix1} \) with row \( \text{rIndex2} \) replaced by the sum of rows \( \text{rIndex1} \) and \( \text{rIndex2} \).
rowDim()

rowDim(\text{Matrix}) \Rightarrow \text{expression}

Returns the number of rows in \text{Matrix}.

Note: See also colDim(), page 17.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} \rightarrow m1
\]

rowDim(m1) \Rightarrow 3

rowNorm()

rowNorm(\text{Matrix}) \Rightarrow \text{expression}

Returns the maximum of the sums of the absolute values of the elements in \text{Matrix}.

Note: All matrix elements must simplify to numbers. See also colNorm(), page 17.

\[
\begin{bmatrix}
-5 & 6 & -7 \\
3 & 4 & 9 \\
9 & -9 & -7
\end{bmatrix}
\]

rowNorm \Rightarrow 25

rowSwap()

rowSwap(\text{Matrix1}, rIndex1, rIndex2) \Rightarrow \text{matrix}

Returns \text{Matrix1} with rows \text{rIndex1} and \text{rIndex2} exchanged.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} \rightarrow mat
\]

rowSwap(mat,1,3) \Rightarrow
\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix}
\]

rref()

rref(\text{Matrix1}, Tol) \Rightarrow \text{matrix}

Returns the reduced row echelon form of \text{Matrix1}.

\[
\begin{bmatrix}
-2 & -2 & 0 & 6 \\
1 & -1 & 9 & 9 \\
-5 & 2 & 4 & -4
\end{bmatrix}
\]

rref \Rightarrow
\[
\begin{bmatrix}
1 & 0 & 0 & \frac{66}{71} \\
0 & 1 & 0 & \frac{147}{71} \\
0 & 0 & 1 & \frac{62}{71}
\end{bmatrix}
\]

Optionally, any matrix element is treated as zero if its absolute value is less than \text{Tol}. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, \text{Tol} is ignored.

- If you use \text{ctrl} \text{enter} or set the \text{Auto} or \text{Approximate} mode to \text{Approximate}, computations are done using floating-point arithmetic.
- If \text{Tol} is omitted or not used, the default tolerance is calculated as:
  \[
  5 \times 10^{-14} \times \max(\text{dim}(	ext{Matrix1})) \times \text{rowNorm}(	ext{Matrix1})
  \]

Note: See also ref(), page 83.
sec()

\[ \text{sec}(\text{Value1}) \Rightarrow \text{value} \]
\[ \text{sec}(\text{List1}) \Rightarrow \text{list} \]

Returns the secant of \( \text{Value1} \) or returns a list containing the secants of all elements in \( \text{List1} \).

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, ′, or to override the angle mode temporarily.

<table>
<thead>
<tr>
<th>Value1</th>
<th>( \text{sec} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>1.41421</td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
<td>{1.00015, 1.00081, 1.00244}</td>
</tr>
</tbody>
</table>

sec^−1()

\[ \text{sec}^\left( \text{Value1} \right) \Rightarrow \text{value} \]
\[ \text{sec}^\left( \text{List1} \right) \Rightarrow \text{list} \]

Returns the angle whose secant is \( \text{Value1} \) or returns a list containing the inverse secants of each element of \( \text{List1} \).

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the keyboard by typing \( \text{arcsec}(\ldots) \).

<table>
<thead>
<tr>
<th>Value1</th>
<th>sec^−1</th>
<th>Value1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>\sqrt{2}</td>
<td>50</td>
<td>{0, 1.0472, 1.36944}</td>
</tr>
<tr>
<td>{1, 2, 5}</td>
<td>{0, 0.648054, 0.198522, 0.036619}</td>
<td></td>
</tr>
</tbody>
</table>

sech()

\[ \text{sech}(\text{Value1}) \Rightarrow \text{value} \]
\[ \text{sech}(\text{List1}) \Rightarrow \text{list} \]

Returns the hyperbolic secant of \( \text{Value1} \) or returns a list containing the hyperbolic secants of the \( \text{List1} \) elements.

<table>
<thead>
<tr>
<th>Value1</th>
<th>sech</th>
<th>List1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.099328</td>
<td>0.648054, 0.198522, 0.036619</td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
<td>{0.648054, 0.198522, 0.036619}</td>
<td></td>
</tr>
</tbody>
</table>

sech^−1()

\[ \text{sech}^\left( \text{Value1} \right) \Rightarrow \text{value} \]
\[ \text{sech}^\left( \text{List1} \right) \Rightarrow \text{list} \]

Returns the inverse hyperbolic secant of \( \text{Value1} \) or returns a list containing the inverse hyperbolic secants of each element of \( \text{List1} \).

**Note:** You can insert this function from the keyboard by typing \( \text{arcsech}(\ldots) \).

<table>
<thead>
<tr>
<th>Value1</th>
<th>sech^−1</th>
<th>Value1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>{0, 2.0944 i, 8.815 + 1.07448 i}</td>
</tr>
<tr>
<td>{-2, 2, 1}</td>
<td>{0, 2.0944 i, 8.815 + 1.07448 i}</td>
<td></td>
</tr>
</tbody>
</table>
seq()

seq(Expr, Var, Low, High[, Step]) → list

Increments Var from Low through High by an increment of Step, evaluates Expr, and returns the results as a list. The original contents of Var are still there after seq() is completed.

The default value for Step = 1.

seqGen()

seqGen(Expr, depVar, (Var0, VarMax)[, ListOfInitTerms [, VarStep [, CeilingValue]]]]) → list

Generates a list of terms for sequence depVar(Var)=Expr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates depVar(Var) for corresponding values of Var using the Expr formula and ListOfInitTerms, and returns the results as a list.

seqGen(ListOrSystemOfExpr, Var, ListOfDepVars, {Var0, VarMax} [, MatrixOfInitTerms [, VarStep [, CeilingValue]]]]) → matrix

Generates a matrix of terms for a system (or list) of sequences ListOfDepVars(Var)=ListOrSystemOfExpr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates ListOfDepVars(Var) for corresponding values of Var using ListOrSystemOfExpr formula and MatrixOfInitTerms, and returns the results as a matrix.

The original contents of Var are unchanged after seqGen() is completed.

The default value for VarStep = 1.

Example in which Var0=2:

System of two sequences:

seqGen\left(\begin{array}{c}
\frac{1}{n^2} - u(n-1) + u(\frac{n(n-1)}{2}) \\
2, 3, 4, 5, 6
\end{array}\right)

Note: The Void (_) in the initial term matrix above is used to indicate that the initial term for u1(n) is calculated using the explicit sequence formula u1(n)=1/n.
seqn()  

`seqn(Expr(u, n [, ListOfInitTerms], nMax [, CeilingValue]]) \rightarrow list`  

Generates a list of terms for a sequence $u(n)=\text{Expr}(u, n)$ as follows: Increments $n$ from 1 through $nMax$ by 1, evaluates $u(n)$ for corresponding values of $n$ using the $\text{Expr}(u, n)$ formula and $\text{ListOfInitTerms}$, and returns the results as a list.  

If $nMax$ is missing, $nMax$ is set to 2500  
If $nMax=0$, $nMax$ is set to 2500  

**Note:** `seqn()` calls `seqGen()` with $n0=1$ and $nstep=1$.

setMode()  

`setMode(modeNameInteger, settingInteger) \rightarrow integer`  

Valid only within a function or program.  

`setMode(list) \rightarrow integer list`  

Display approximate value of $\pi$ using the default setting for Display Digits, and then display $\pi$ with a setting of Fix2. Check to see that the default is restored after the program executes.  

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td><code>$1=\text{Float}, 2=\text{Float1}, 3=\text{Float2}, 4=\text{Float3}, 5=\text{Float4}, 6=\text{Float5}, 7=\text{Float6}, 8=\text{Float7}, 9=\text{Float8}, 10=\text{Float9}, 11=\text{Float10}, 12=\text{Float11}, 13=\text{Float12}, 14=\text{Fix0}, 15=\text{Fix1}, 16=\text{Fix2}, 17=\text{Fix3}, 18=\text{Fix4}, 19=\text{Fix5}, 20=\text{Fix6}, 21=\text{Fix7}, 22=\text{Fix8}, 23=\text{Fix9}, 24=\text{Fix10}, 25=\text{Fix11}, 26=\text{Fix12}</code></td>
</tr>
<tr>
<td>Angle</td>
<td>2</td>
<td><code>$1=\text{Radian}, 2=\text{Degree}, 3=\text{Gradian}</code></td>
</tr>
</tbody>
</table>
shift()  

**shift(Integer1[,#ofShifts]) ⇒ integer**  
Shifts the bits in a binary integer. You can enter `Integer1` in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of `Integer1` is too large for this form, a symmetric modulo operation brings it within the range. For more information, see **Base2**, page 12.

If `#ofShifts` is positive, the shift is to the left. If `#ofShifts` is negative, the shift is to the right. The default is -1 (shift right one bit).

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

For example, in a right shift:

Each bit shifts right.

```plaintext
0b0000000000000111101011000011010
```

Inserts 0 if leftmost bit is 0, or 1 if leftmost bit is 1.

```
0b00000000000000111101011000011010
```

The result is displayed according to the Base mode. Leading zeros are not shown.

**shift(List1 [,#ofShifts]) ⇒ list**  
Returns a copy of `List1` shifted right or left by `#ofShifts` elements. Does not alter `List1`.

If `#ofShifts` is positive, the shift is to the left. If `#ofShifts` is negative, the shift is to the right. The default is -1 (shift right one element).

Elements introduced at the beginning or end of `list` by the shift are set to the symbol "undef".

**shift(String1 [,#ofShifts]) ⇒ string**  
Returns a copy of `String1` shifted right or left by `#ofShifts` characters. Does not alter `String1`.

If `#ofShifts` is positive, the shift is to the left. If `#ofShifts` is negative, the shift is to the right. The default is -1 (shift right one character).

Characters introduced at the beginning or end of `string` by the shift are set to a space.
sign()  

sign(Value1) ⇒ value
sign(List1) ⇒ list
sign(Matrix1) ⇒ matrix

For real and complex Value1, returns Value1/abs(Value1) when Value1 ≠ 0.

Returns 1 if Value1 is positive.
Returns -1 if Value1 is negative.

sign(0) returns ±1 if the complex format mode is Real; otherwise, it returns itself.

sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

simult()  

simult(coeffMatrix, constVector, Tol) ⇒ matrix

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also linSolve(), page 55.

coeffMatrix must be a square matrix that contains the coefficients of the equations.

constVector must have the same number of rows (same dimension) as coeffMatrix and contain the constants.

Optionally, any matrix element is treated as zero if its absolute value is less than Tol. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, Tol is ignored.

- If you set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as: 5E-14 · max(dim(coeffMatrix)) · rowNorm(coeffMatrix)

Solve for x and y:

\[
\begin{align*}
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
x + 2y &= 1 \\
3x + 4y &= -1 \\
\end{align*}
\]

simult(\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\]
, \[
\begin{bmatrix}
1 \\
1 \\
\end{bmatrix}
\]
, -1)

\[
\begin{align*}
x &= -3 \\
y &= 2 \\
\end{align*}
\]

The solution is x=-3 and y=2.

Solve:

\[
\begin{align*}
ax + by &= 1 \\
\end{align*}
\]

\[
\begin{align*}
ax + dy &= 2 \\
\end{align*}
\]

simult(coeffMatrix, constMatrix, Tol) ⇒ matrix

Solves multiple systems of linear equations, where each system has the same equation coefficients but different constants.

Each column in constMatrix must contain the constants for a system of equations. Each column in the resulting matrix contains the solution for the corresponding system.

Solve:

\[
\begin{align*}
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
x + 2y &= 1 \\
3x + 4y &= -1 \\
\end{align*}
\]

\[
\begin{align*}
x + 2y &= 2 \\
3x + 4y &= -3 \\
\end{align*}
\]

simult(\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\]
, \[
\begin{bmatrix}
1 & 2 \\
1 & -3 \\
\end{bmatrix}
\]
, -3)

\[
\begin{align*}
x &= 3 \\
y &= 9/2 \\
\end{align*}
\]

For the first system, x=3 and y=9/2. For the second system, x=-7 and y=9/2.
**sin()**

\[
sin(\text{Value1}) \Rightarrow \text{value} \\
sin(\text{List1}) \Rightarrow \text{list}
\]

- **sin(\text{Value1})** returns the sine of the argument.
- **sin(\text{List1})** returns a list of the sines of all elements in List1.

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, ‰, or ′ to override the angle mode setting temporarily.

### In Degree angle mode:

\[
\sin\left(\frac{\pi}{4}\right) = 0.707107 \\
\sin(45) = 0.707107 \\
\sin\left\{0.60,90\right\} = \{0.866025,1.\}
\]

### In Gradian angle mode:

\[
\sin(50) = 0.707107
\]

### In Radian angle mode:

\[
\sin\left(\frac{\pi}{4}\right) = 0.707107 \\
\sin(45°) = 0.707107
\]

**sin(squareMatrix1)** \(\Rightarrow\) squareMatrix

- **sin(squareMatrix1)** returns the matrix sine of squareMatrix1. This is not the same as calculating the sine of each element. For information about the calculation method, refer to **cos()**.

- **squareMatrix1** must be diagonalizable. The result always contains floating-point numbers.

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.9424 & -0.04542 & -0.031999 \\
-0.04542 & 0.949254 & -0.020274 \\
-0.048739 & -0.00523 & 0.961051
\end{bmatrix}
\]

### sin¹()

- **sin¹(\text{Value1})** \(\Rightarrow\) value
- **sin¹(\text{List1})** \(\Rightarrow\) list

- **sin¹(\text{Value1})** returns the angle whose sine is Value1.
- **sin¹(\text{List1})** returns a list of the inverse sines of each element of List1.

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the keyboard by typing arcsin(...).

**sin¹(squareMatrix1)** \(\Rightarrow\) squareMatrix

- **sin¹(squareMatrix1)** returns the matrix inverse sine of squareMatrix1. This is not the same as calculating the inverse sine of each element. For information about the calculation method, refer to **cos()**.

- **squareMatrix1** must be diagonalizable. The result always contains floating-point numbers.

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
-0.164058 - 0.064606\cdot i & 1.49086 - 2.1051\cdot i \\
0.725533 - 1.51594\cdot i & 0.947305 - 0.7783\cdot i \\
2.08316 - 2.63205\cdot i & -1.79018 + 1.2718\cdot i
\end{bmatrix}
\]

To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.
sinh()

\[
\sinh(\text{Numver1}) \Rightarrow \text{value} \\
\sinh(\text{List1}) \Rightarrow \text{list} \\
\sinh(\text{Value1}) \rightarrow \text{value} \\
\sinh(\text{List1}) \rightarrow \text{list}
\]

\(\sinh(\text{Value1})\) returns the hyperbolic sine of the argument.

\(\sinh(\text{List1})\) returns a list of the hyperbolic sines of each element of \(\text{List1}\).

\(\sinh(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\)

Returns the matrix hyperbolic sine of \(\text{squareMatrix1}\). This is not the same as calculating the hyperbolic sine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

| \(\sinh(1.2)\) | 1.50946 |
| \(\sinh(\{0,1.2,3\})\) | \(\{0,1.50946,10.0179\}\) |

In Radian angle mode:

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\]

\[
\begin{array}{ccc}
360.954 & 305.708 & 239.604 \\
352.912 & 233.495 & 193.564 \\
298.632 & 154.599 & 140.251
\end{array}
\]

sinh^−1()

\[
\sinh^−1(\text{Value1}) \Rightarrow \text{value} \\
\sinh^−1(\text{List1}) \Rightarrow \text{list} \\
\sinh^−1(\text{Value1}) \rightarrow \text{value} \\
\sinh^−1(\text{List1}) \rightarrow \text{list}
\]

\(\sinh^−1(\text{Value1})\) returns the inverse hyperbolic sine of the argument.

\(\sinh^−1(\text{List1})\) returns a list of the inverse hyperbolic sines of each element of \(\text{List1}\).

Note: You can insert this function from the keyboard by typing \(\text{arcsinh(\ldots)}\).

\(\sinh^−1(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\)

Returns the matrix inverse hyperbolic sine of \(\text{squareMatrix1}\). This is not the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

| \(\sinh^−1(0)\) | 0 |
| \(\sinh^−1(\{0,2,1,3\})\) | \(\{0,1.48748,1.81845\}\) |

In Radian angle mode:

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\]

\[
\begin{array}{ccc}
0.041751 & 2.15557 & 1.1582 \\
1.46382 & 0.926568 & 0.112557 \\
2.75079 & -1.5283 & 0.57268
\end{array}
\]
SinReg

SinReg \( X, Y[,\text{[Iterations]},[\text{Period}],[\text{Category},\text{Include}]] \)

Computes the sinusoidal regression on lists \( X \) and \( Y \). A summary of results is stored in the \textit{stat.results} variable. (See page 98.)

All the lists must have equal dimension except for \textit{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\textit{Iterations} is a value that specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

\textit{Period} specifies an estimated period. If omitted, the difference between values in \( X \) should be equal and in sequential order. If you specify \textit{Period}, the differences between \( x \) values can be unequal.

\textit{Category} is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\textit{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

The output of SinReg is always in radians, regardless of the angle mode setting.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{stat.RegEqn}</td>
<td>Regression Equation: ( a \cdot \sin(bx+c)+d )</td>
</tr>
<tr>
<td>\textit{stat.a}, \textit{stat.b}, \textit{stat.c}, \textit{stat.d}</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>\textit{stat.Resid}</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>\textit{stat.XReg}</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of \textit{Freq}, \textit{Category List}, and \textit{Include Categories}</td>
</tr>
<tr>
<td>\textit{stat.YReg}</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of \textit{Freq}, \textit{Category List}, and \textit{Include Categories}</td>
</tr>
<tr>
<td>\textit{stat.FreqReg}</td>
<td>List of frequencies corresponding to \textit{stat.XReg} and \textit{stat.YReg}</td>
</tr>
</tbody>
</table>

SortA

SortA \( \text{List1}, \text{List2}[,\text{List3}] \ldots \)

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 132.

\( \{2,1,4,3\} \rightarrow \text{list1} \) \( \{2,1,4,3\} \)

\( \text{SortA list1} \) \( \text{Done} \)

\( list1 \) \( \{1,2,3,4\} \)

\( \{4,3,2,1\} \rightarrow \text{list2} \) \( \{4,3,2,1\} \)

\( \text{SortA list2, list1} \) \( \text{Done} \)

\( list2 \) \( \{1,2,3,4\} \)

\( list1 \) \( \{4,3,2,1\} \)
SortD

SortD \( List1, List2, List3, \ldots \)
SortD \( Vector1, Vector2, Vector3, \ldots \)

Identical to SortA, except SortD sorts the elements in descending order.

Empty (void) elements within the first argument move to the bottom.
For more information on empty elements, see page 132.

\[ \{2, 1, 4, 3\} \rightarrow list1 \]
\[ \{1, 2, 3, 4\} \rightarrow list2 \]

SortD \( list1, list2 \)
\[ lavish \]

\[ list1 \]
\[ \{4, 3, 2, 1\} \]
\[ list2 \]
\[ \{3, 4, 1, 2\} \]

\( \triangledown \) Sphere

Vector \( \triangledown \) Sphere

Note: You can insert this operator from the computer keyboard by typing \( @> \) Sphere.

Displays the row or column vector in spherical form \( [\rho \angle \theta \angle \phi] \).

Vector must be of dimension 3 and can be either a row or a column vector.

Note: \( \triangledown \) Sphere is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

\[ [1 \ 2 \ 3] \triangledown \text{Sphere} \]
\[ \begin{bmatrix} 3.74166 \ & 1.10715 \ & 0.640522 \end{bmatrix} \]

\[ \left( 2 \ \angle \frac{\pi}{4} \ 3 \right) \triangledown \text{Sphere} \]
\[ \begin{bmatrix} 3.60555 \ & 0.785398 \ & 0.588003 \end{bmatrix} \]

sqrt()

sqrt(\( Value1 \)) \( \Rightarrow \) value
sqrt(\( List1 \)) \( \Rightarrow \) list

Returns the square root of the argument.

For a list, returns the square roots of all the elements in \( List1 \).

Note: See also Square root template, page 1.
stat.results

Displays results from a statistics calculation.

The results are displayed as a set of name-value pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

**Note:** Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.a</td>
<td></td>
</tr>
<tr>
<td>stat.AdjR²</td>
<td></td>
</tr>
<tr>
<td>stat.b</td>
<td></td>
</tr>
<tr>
<td>stat.b0</td>
<td></td>
</tr>
<tr>
<td>stat.b1</td>
<td></td>
</tr>
<tr>
<td>stat.b2</td>
<td></td>
</tr>
<tr>
<td>stat.b3</td>
<td></td>
</tr>
<tr>
<td>stat.b4</td>
<td></td>
</tr>
<tr>
<td>stat.b5</td>
<td></td>
</tr>
<tr>
<td>stat.b6</td>
<td></td>
</tr>
<tr>
<td>stat.b7</td>
<td></td>
</tr>
<tr>
<td>stat.b8</td>
<td></td>
</tr>
<tr>
<td>stat.b9</td>
<td></td>
</tr>
<tr>
<td>stat.b10</td>
<td></td>
</tr>
<tr>
<td>stat.blst</td>
<td></td>
</tr>
<tr>
<td>stat.χ²</td>
<td></td>
</tr>
<tr>
<td>stat.c</td>
<td></td>
</tr>
<tr>
<td>stat.Clower</td>
<td></td>
</tr>
<tr>
<td>stat.ClowerList</td>
<td></td>
</tr>
<tr>
<td>stat.ComplList</td>
<td></td>
</tr>
<tr>
<td>stat.Compmatrix</td>
<td></td>
</tr>
<tr>
<td>stat.CookDist</td>
<td></td>
</tr>
<tr>
<td>stat.CUpper</td>
<td></td>
</tr>
<tr>
<td>stat.CUpperList</td>
<td></td>
</tr>
<tr>
<td>stat.d</td>
<td></td>
</tr>
<tr>
<td>stat.dDenom</td>
<td></td>
</tr>
<tr>
<td>stat.dBlock</td>
<td></td>
</tr>
<tr>
<td>stat.dCol</td>
<td></td>
</tr>
<tr>
<td>stat.dError</td>
<td></td>
</tr>
<tr>
<td>stat.dInteract</td>
<td></td>
</tr>
<tr>
<td>stat.dReg</td>
<td></td>
</tr>
<tr>
<td>stat.dNúmer</td>
<td></td>
</tr>
<tr>
<td>stat.dRow</td>
<td></td>
</tr>
<tr>
<td>stat.DW</td>
<td></td>
</tr>
<tr>
<td>stat.e</td>
<td></td>
</tr>
<tr>
<td>stat.ExpMatrix</td>
<td></td>
</tr>
<tr>
<td>stat.F</td>
<td></td>
</tr>
<tr>
<td>stat.FBlock</td>
<td></td>
</tr>
<tr>
<td>stat.FCol</td>
<td></td>
</tr>
<tr>
<td>stat.FInteract</td>
<td></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td></td>
</tr>
<tr>
<td>stat.FRow</td>
<td></td>
</tr>
<tr>
<td>stat.Fr</td>
<td></td>
</tr>
<tr>
<td>stat.Leverage</td>
<td></td>
</tr>
<tr>
<td>stat.LowerPred</td>
<td></td>
</tr>
<tr>
<td>stat.LowerVal</td>
<td></td>
</tr>
<tr>
<td>stat.m</td>
<td></td>
</tr>
<tr>
<td>stat.MaxX</td>
<td></td>
</tr>
<tr>
<td>stat.MaxY</td>
<td></td>
</tr>
<tr>
<td>stat.ME</td>
<td></td>
</tr>
<tr>
<td>stat.MedianX</td>
<td></td>
</tr>
<tr>
<td>stat.MEDy</td>
<td></td>
</tr>
<tr>
<td>stat.MEPred</td>
<td></td>
</tr>
<tr>
<td>stat.MinX</td>
<td></td>
</tr>
<tr>
<td>stat.MinY</td>
<td></td>
</tr>
<tr>
<td>stat.MS</td>
<td></td>
</tr>
<tr>
<td>stat.MSBlock</td>
<td></td>
</tr>
<tr>
<td>stat.MSCol</td>
<td></td>
</tr>
<tr>
<td>stat.MSError</td>
<td></td>
</tr>
<tr>
<td>stat.MSInteract</td>
<td></td>
</tr>
<tr>
<td>stat.MSReg</td>
<td></td>
</tr>
<tr>
<td>stat.MSRow</td>
<td></td>
</tr>
<tr>
<td>stat.n</td>
<td></td>
</tr>
<tr>
<td>stat.β</td>
<td></td>
</tr>
<tr>
<td>stat.β1</td>
<td></td>
</tr>
<tr>
<td>stat.β2</td>
<td></td>
</tr>
<tr>
<td>stat.βDiff</td>
<td></td>
</tr>
<tr>
<td>stat.PLst</td>
<td></td>
</tr>
<tr>
<td>stat.PVal</td>
<td></td>
</tr>
<tr>
<td>stat.PValBlock</td>
<td></td>
</tr>
<tr>
<td>stat.PValCol</td>
<td></td>
</tr>
<tr>
<td>stat.PValInteract</td>
<td></td>
</tr>
<tr>
<td>stat.PValRow</td>
<td></td>
</tr>
<tr>
<td>stat.Q1X</td>
<td></td>
</tr>
<tr>
<td>stat.Q1Y</td>
<td></td>
</tr>
<tr>
<td>stat.Q3X</td>
<td></td>
</tr>
<tr>
<td>stat.Q3Y</td>
<td></td>
</tr>
<tr>
<td>stat.ρ</td>
<td></td>
</tr>
<tr>
<td>stat.RegEqn</td>
<td></td>
</tr>
<tr>
<td>stat.Resid</td>
<td></td>
</tr>
<tr>
<td>stat.ResidTrans</td>
<td></td>
</tr>
<tr>
<td>stat.ox</td>
<td></td>
</tr>
<tr>
<td>stat.ox1</td>
<td></td>
</tr>
<tr>
<td>stat.ox2</td>
<td></td>
</tr>
<tr>
<td>stat.Sx</td>
<td></td>
</tr>
<tr>
<td>stat.Sx²</td>
<td></td>
</tr>
<tr>
<td>stat.Σxy</td>
<td></td>
</tr>
<tr>
<td>stat.Σy</td>
<td></td>
</tr>
<tr>
<td>stat.Σy²</td>
<td></td>
</tr>
<tr>
<td>stat.s</td>
<td></td>
</tr>
<tr>
<td>stat.S</td>
<td></td>
</tr>
<tr>
<td>stat.SError</td>
<td></td>
</tr>
<tr>
<td>stat.SReg</td>
<td></td>
</tr>
<tr>
<td>stat.SInteract</td>
<td></td>
</tr>
<tr>
<td>stat.SList</td>
<td></td>
</tr>
<tr>
<td>stat.X</td>
<td></td>
</tr>
<tr>
<td>stat.X1</td>
<td></td>
</tr>
<tr>
<td>stat.X2</td>
<td></td>
</tr>
<tr>
<td>stat.XDiff</td>
<td></td>
</tr>
<tr>
<td>stat.XList</td>
<td></td>
</tr>
<tr>
<td>stat.XReg</td>
<td></td>
</tr>
<tr>
<td>stat.XVal</td>
<td></td>
</tr>
<tr>
<td>stat.XValList</td>
<td></td>
</tr>
<tr>
<td>stat.y</td>
<td></td>
</tr>
<tr>
<td>stat.Y</td>
<td></td>
</tr>
<tr>
<td>stat.YList</td>
<td></td>
</tr>
<tr>
<td>stat.YReg</td>
<td></td>
</tr>
<tr>
<td>stat.SSBlock</td>
<td></td>
</tr>
<tr>
<td>stat.SSCol</td>
<td></td>
</tr>
<tr>
<td>stat.SSX</td>
<td></td>
</tr>
<tr>
<td>stat.SSY</td>
<td></td>
</tr>
<tr>
<td>stat.SSReg</td>
<td></td>
</tr>
<tr>
<td>stat.SSInteract</td>
<td></td>
</tr>
<tr>
<td>stat.SSRow</td>
<td></td>
</tr>
<tr>
<td>stat.t</td>
<td></td>
</tr>
<tr>
<td>stat.t1</td>
<td></td>
</tr>
<tr>
<td>stat.t2</td>
<td></td>
</tr>
<tr>
<td>stat.tList</td>
<td></td>
</tr>
<tr>
<td>stat.UPPred</td>
<td></td>
</tr>
<tr>
<td>stat.UPVal</td>
<td></td>
</tr>
<tr>
<td>stat.ω</td>
<td></td>
</tr>
<tr>
<td>stat.ω1</td>
<td></td>
</tr>
<tr>
<td>stat.ω2</td>
<td></td>
</tr>
<tr>
<td>stat.ωDiff</td>
<td></td>
</tr>
<tr>
<td>stat.ωList</td>
<td></td>
</tr>
<tr>
<td>stat.ωReg</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Each time the lists & Spreadsheet application calculates statistical results, it copies the "stat." group variables to a "stat#." group, where # is a number that is incremented automatically. This lets you maintain previous results while performing multiple calculations.
**stat.values**

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike **stat.results**, **stat.values** omits the names associated with the values.

You can copy a value and paste it into other locations.

**stDevPop()**

\[
\text{stDevPop}(\text{List}, \text{freqList}) \Rightarrow \text{expression}
\]

Returns the population standard deviation of the elements in \( \text{List} \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List} \).

**Note:** \( \text{List} \) must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 132.

**stDevPop\text{Matrix1}, \text{freqMatrix}) \Rightarrow \text{matrix}**

Returns a row vector of the population standard deviations of the columns in \( \text{Matrix1} \).

Each \( \text{freqMatrix} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{Matrix1} \).

**Note:** \( \text{Matrix1} \) must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 132.

**stDevSamp()**

\[
\text{stDevSamp}(\text{List}, \text{freqList}) \Rightarrow \text{expression}
\]

Returns the sample standard deviation of the elements in \( \text{List} \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List} \).

**Note:** \( \text{List} \) must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 132.

**stDevSamp\text{Matrix1}, \text{freqMatrix}) \Rightarrow \text{matrix}**

Returns a row vector of the sample standard deviations of the columns in \( \text{Matrix1} \).

Each \( \text{freqMatrix} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{Matrix1} \).

**Note:** \( \text{Matrix1} \) must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 132.
Stop

Programming command: Terminates the program.

Stop is not allowed in functions.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing [→] instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

\[
\begin{align*}
\text{i:=0} & \quad 0 \\
\text{Define prog1()=Pgm} & \\
& \text{For i,1,10,1} \\
& \text{If i=5} \\
& \text{Stop} \\
& \text{EndFor} \\
& \text{EndPgm}
\end{align*}
\]

\[
\begin{array}{l}
\text{prog1()} \quad \text{Done} \\
\text{i} \quad 5
\end{array}
\]

Store

See → (store), page 130.

string()

\[
\text{string(Expr) \Rightarrow string}
\]

Simplifies Expr and returns the result as a character string.

\[
\text{string(1.2345)} \quad \text{"1.2345"}
\]

\[
\text{string(1+2)} \quad \text{"3"}
\]

subMat()

\[
\text{subMat(Matrix1[, startRow] [, startCol] [, endRow] [, endCol])} \Rightarrow \text{matrix}
\]

Returns the specified submatrix of Matrix1.

Defaults: startRow=1, startCol=1, endRow=last row, endCol=last column.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix} \Rightarrow m1
\]

\[
\begin{array}{c}
\text{subMat(m1,2,1,3,2)} \\
\text{subMat(m1,2,2)}
\end{array}
\]

Sum (Sigma)

See Σ(), page 126.

sum()

\[
\text{sum(List[, Start[, End]])} \Rightarrow \text{expression}
\]

Returns the sum of all elements in List.

Start and End are optional. They specify a range of elements.

Any void argument produces a void result. Empty (void) elements in List are ignored. For more information on empty elements, see page 132.

\[
\begin{align*}
\text{sum\{1,2,3,4,5\}} & \quad 15 \\
\text{sum\{a,2\cdot a,3\cdot a\}} & \quad \text{"Error: Variable is not defined"} \\
\text{sum\{seq(n,n,1,10)\}} & \quad 55 \\
\text{sum\{1,3,5,7,9\},3} & \quad 21
\end{align*}
\]
sum() → matrix

sum(Matrix1[, Start[, End]])

Returns a row vector containing the sums of all elements in the columns in Matrix1.

Start and End are optional. They specify a range of rows.

Any void argument produces a void result. Empty (void) elements in Matrix1 are ignored. For more information on empty elements, see page 132.

sum([1 2 3]
    [4 5 6])

\[ \begin{pmatrix} 5 & 7 & 9 \end{pmatrix} \]

sum([1 2 3]
    [4 5 6]
    [7 8 9])

\[ \begin{pmatrix} 12 & 15 & 18 \end{pmatrix} \]

sum([1 2 3]
    [4 5 6]
    [7 8 9])

\[ \begin{pmatrix} 11 & 13 & 15 \end{pmatrix} \]

sumIf() → value

sumIf(List, Criteria[, SumList])

Returns the accumulated sum of all elements in List that meet the specified Criteria. Optionally, you can specify an alternate list, SumList, to supply the elements to accumulate.

List can be an expression, list, or matrix. SumList, if specified, must have the same dimension(s) as List.

Criteria can be:

- A value, expression, or string. For example, 34 accumulates only those elements in List that simplify to the value 34.
- A Boolean expression containing the symbol ? as a placeholder for each element. For example, \( 10 \? < 10 \) accumulates only those elements in List that are less than 10.

When a List element meets the Criteria, the element is added to the accumulating sum. If you include SumList, the corresponding element from SumList is added to the sum instead.

Within the Lists & Spreadsheet application, you can use a range of cells in place of List and SumList.

Empty (void) elements are ignored. For more information on empty elements, see page 132.

Note: See also countIf(), page 23.

sumSeq() → See Σ₀, page 126.

system() → list

system(Value1[, Value2[, Value3[, ...]]])

Returns a system of equations, formatted as a list. You can also create a system by using a template.

\[ T \]

T (transpose) → matrix

\[ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \]

\[ \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \]
\( \tan() \)

**\( \tan(\text{Value1}) \Rightarrow \text{value} \)**

**\( \tan(\text{List1}) \Rightarrow \text{list} \)**

\( \tan(\text{Value1}) \) returns the tangent of the argument.

\( \tan(\text{List1}) \) returns a list of the tangents of all elements in \( \text{List1} \).

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, ′ or ″ to override the angle mode setting temporarily.

\( \tan(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

Returns the matrix tangent of \( \text{squareMatrix1} \). This is not the same as calculating the tangent of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

\( \tan^\prime() \)

**\( \tan^\prime(\text{Value1}) \Rightarrow \text{value} \)**

**\( \tan^\prime(\text{List1}) \Rightarrow \text{list} \)**

\( \tan^\prime(\text{Value1}) \) returns the angle whose tangent is \( \text{Value1} \).

\( \tan^\prime(\text{List1}) \) returns a list of the inverse tangents of each element of \( \text{List1} \).

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the keyboard by typing \( \text{arctan(...)} \).
\( \tan^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

Returns the matrix inverse tangent of \( \text{squareMatrix1} \). This is not the same as calculating the inverse tangent of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
-0.083658 & 1.26629 & 0.62263 \\
0.748539 & 0.630015 & -0.070012 \\
1.68608 & -1.18244 & 0.455126
\end{bmatrix}
\]

\( \tanh() \)

\( \tanh(\text{Value1}) \Rightarrow \text{value} \)

\( \tanh(\text{List1}) \Rightarrow \text{list} \)

\( \tanh(\text{Value1}) \) returns the hyperbolic tangent of the argument.

\( \tanh(\text{List1}) \) returns a list of the hyperbolic tangents of each element of \( \text{List1} \).

\( \tanh(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

Returns the matrix hyperbolic tangent of \( \text{squareMatrix1} \). This is not the same as calculating the hyperbolic tangent of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
-0.097966 & 0.933436 & 0.425972 \\
0.488147 & 0.538881 & -0.129382 \\
1.28295 & -1.03425 & 0.428817
\end{bmatrix}
\]

\( \tanh^{-1}() \)

\( \tanh^{-1}(\text{Value1}) \Rightarrow \text{value} \)

\( \tanh^{-1}(\text{List1}) \Rightarrow \text{list} \)

\( \tanh^{-1}(\text{Value1}) \) returns the inverse hyperbolic tangent of the argument.

\( \tanh^{-1}(\text{List1}) \) returns a list of the inverse hyperbolic tangents of each element of \( \text{List1} \).

\( \text{Note:} \) You can insert this function from the keyboard by typing \( \text{arctanh}(\ldots) \).

\( \tanh^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

Returns the matrix inverse hyperbolic tangent of \( \text{squareMatrix1} \). This is not the same as calculating the inverse hyperbolic tangent of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

In Rectangular complex format:

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0.099353+0.164058\cdot i & 0.267834-1.4908\cdot i \\
-0.087596-0.725533\cdot i & 0.479679-0.94730\cdot i \\
0.511463-2.08316\cdot i & -0.878563+1.7901\cdot i
\end{bmatrix}
\]

To see the entire result, press \( \text{up} \) and then use \( \text{left} \) and \( \text{right} \) to move the cursor.
tCdf() \[ \text{Catalog > tCdf(lowBound,upBound,df)} \Rightarrow \text{number if lowBound and upBound are numbers, list if lowBound and upBound are lists} \]

Computes the Student-\(t\) distribution probability between \(\text{lowBound}\) and \(\text{upBound}\) for the specified degrees of freedom \(\text{df}\).

For \(P(X \leq \text{upBound})\), set \(\text{lowBound} = -9E999\).

Text \[ \text{Catalog > Text} \]

Text \(\text{promptString [, DispFlag]}\)

Programming command: Pauses the program and displays the character string \(\text{promptString}\) in a dialog box.

When the user selects \text{OK}, program execution continues.

The optional \(\text{flag}\) argument can be any expression.

• If \(\text{DispFlag}\) is omitted or evaluates to \(1\), the text message is added to the Calculator history.
• If \(\text{DispFlag}\) evaluates to \(0\), the text message is not added to the history.

If the program needs a typed response from the user, refer to \text{Request}, page 84, or \text{RequestStr}, page 85.

\textbf{Note:} You can use this command within a user-defined program but not within a function.

Define a program that pauses to display each of five random numbers in a dialog box.

Within the \text{Prgm...EndPrgm} template, complete each line by pressing \(\text{↵} \) instead of \text{enter}. On the computer keyboard, hold down \text{Alt} and press \text{Enter}.

Define \text{text_demo()=Prgm}

For \(i,1,5\)

\text{strinfo:="Random number “ & string(rand(i))}

\text{Text strinfo}

EndFor

EndPrgm

Run the program:

\text{text_demo()}

Sample of one dialog box:

```
Random number (0.943597)
OK
```

Then \[ \text{See if, page 45.} \]

tInterval \[ \text{Catalog > tInterval} \]

\text{tInterval List[,Freq[,CLevel]]} \quad \text{(Data list input)}

\text{tInterval } \overline{x},sx,n[,CLevel] \quad \text{(Summary stats input)}

Computes a \(t\) confidence interval. A summary of results is stored in the \text{stat.results} variable. (See page 98.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Output variable} & \textbf{Description} \\
\hline
\text{stat.CLower, stat.CUpper} & Confidence interval for an unknown population mean \\
\hline
\text{stat.}\overline{x} & Sample mean of the data sequence from the normal random distribution \\
\hline
\text{stat.ME} & Margin of error \\
\hline
\text{stat.df} & Degrees of freedom \\
\hline
\text{stat.sx} & Sample standard deviation \\
\hline
\end{tabular}
\end{center}
Output variable | Description
---|---
stat.n | Length of the data sequence with sample mean

tInterval_2Samp

**tInterval_2Samp**

List1,List2,Freq1[,Freq2[,CLevel[,Pooled]]]]

(Data list input)

**tInterval_2Samp** x1,ssx1,n1,x2,ssx2,n2[,CLevel[,Pooled]]]

(Summary stats input)

Computes a two-sample t confidence interval. A summary of results is stored in the `stat.results` variable. (See page 98.)

Pooled=1 pools variances; Pooled=0 does not pool variances.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>stat.x1-x2</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.x1, stat.x2</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>stat.sp</td>
<td>The pooled standard deviation. Calculated when Pooled = YES</td>
</tr>
</tbody>
</table>

tPdf()

**tPdf(XVal,df)** ⇒ number if XVal is a number, list if XVal is a list

Computes the probability density function (pdf) for the Student-t distribution at a specified x value with specified degrees of freedom df.

trace()

**trace(squareMatrix)** ⇒ value

Returns the trace (sum of all the elements on the main diagonal) of squareMatrix.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>trace(\begin{pmatrix} 1 &amp; 2 &amp; 3 \ 4 &amp; 5 &amp; 6 \ 7 &amp; 8 &amp; 9 \end{pmatrix})</td>
<td>15</td>
</tr>
<tr>
<td>trace(\begin{pmatrix} a &amp; 0 \ 1 &amp; a \end{pmatrix})</td>
<td>24</td>
</tr>
<tr>
<td>a:=12</td>
<td>12</td>
</tr>
</tbody>
</table>
Try
block1
Else
block2
EndTry

Executes block1 unless an error occurs. Program execution transfers to block2 if an error occurs in block1. System variable errCode contains the error code to allow the program to perform error recovery. For a list of error codes, see "Error codes and messages," page 138.

block1 and block2 can be either a single statement or a series of statements separated with the "::" character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Example 2
To see the commands Try, ClrErr, and PassErr in operation, enter the eigenvals() program shown at the right. Run the program by executing each of the following expressions.

Define eigenvals(a,b)=Prgm
Try
Disp "A= ",a
Disp "B= ",b
Disp " 
Disp "Eigenvalues of A·B are:",eigVl(a*b)
Else
If errCode=230 Then
Disp "Error: Product of A·B must be a square matrix"
ClrErr
Else
PassErr
EndIf
EndTry
EndPrgm

Define prog1()=Prgm
Try
z::z+1
Disp "z incremented."
Else
Disp "Sorry, z undefined."
EndTry
EndPrgm

Example 2
To see the commands Try, ClrErr, and PassErr in operation, enter the eigenvals() program shown at the right. Run the program by executing each of the following expressions.

Define eigenvals(a,b)=Prgm
Try
Disp "A= ",a
Disp "B= ",b
Disp " 
Disp "Eigenvalues of A·B are:",eigVl(a*b)
Else
If errCode=230 Then
Disp "Error: Product of A·B must be a square matrix"
ClrErr
Else
PassErr
EndIf
EndTry
EndPrgm

Define eigenvals(a,b)=Prgm
Try
Disp "A= ",a
Disp "B= ",b
Disp " 
Disp "Eigenvalues of A·B are:",eigVl(a*b)
Else
If errCode=230 Then
Disp "Error: Product of A·B must be a square matrix"
ClrErr
Else
PassErr
EndIf
EndTry
EndPrgm

Note: See also ClrErr, page 17, and PassErr, page 74.

\[
eigenvals = \begin{bmatrix} -3 \\ -41 \\ 5 \end{bmatrix}
\]

\[
= \begin{bmatrix} 2 & -3.1 \end{bmatrix}
\]

Note: See also ClrErr, page 17, and PassErr, page 74.

(tTest)

\(\text{tTest} \mu_0,\text{List}[,\text{Freq}[,\text{Hypoth}]]\)

(Data list input)

\(\text{tTest} \mu_0,\bar{x},s,n[,\text{Hypoth}]\)

(Summary stats input)

Performs a hypothesis test for a single unknown population mean \(\mu\) when the population standard deviation \(\sigma\) is unknown. A summary of results is stored in the \(\text{stat.results}\) variable. (See page 98.)

Test \(H_0: \mu = \mu_0\), against one of the following:

For \(H_a: \mu < \mu_0\), set \text{Hypoth}<0
For \(H_a: \mu \neq \mu_0\) (default), set \text{Hypoth}=0
For \(H_a: \mu > \mu_0\), set \text{Hypoth}>0

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.t</td>
<td>$(\bar{x} - \mu_0) / (\text{stdev} / \sqrt{n})$</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.(\bar{x})</td>
<td>Sample mean of the data sequence in List</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of the data sequence</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

### tTest_2Samp

**tTest_2Samp** `List1,List2[,Freq1[,Freq2[,Hypoth[,Pooled]]]]`

(Data list input)

**tTest_2Samp** `\(\bar{x},s_1,n_1,\bar{x},s_2,n_2[,Hypoth[,Pooled]]\)`

(Summary stats input)

Computes a two-sample \(t\) test. A summary of results is stored in the `stat.results` variable. (See page 98.)

Test \(H_0: \mu_1 = \mu_2\), against one of the following:

- For \(H_a: \mu_1 < \mu_2\), set `Hypoth<0`
- For \(H_a: \mu_1 \neq \mu_2\) (default), set `Hypoth=0`
- For \(H_a: \mu_1 > \mu_2\), set `Hypoth>0`

`Pooled=1` pools variances

`Pooled=0` does not pool variances

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

### tvmFV()

**tvmFV** `\(N,I,PV,Pmt,PpY,CpY,PmtAt\) \Rightarrow \text{value}`

Financial function that calculates the future value of money.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 108. See also `amortTbl()`, page 6.
These time-value-of-money argument names are similar to the TVM variable names (such as tvm.pv and tvm.pmt) that are used by the Calculator application’s finance solver. Financial functions, however, do not store their argument values or results to the TVM variables.

Financial function that calculates the interest rate per year.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 108. See also amortTbl(), page 6.

Financial function that calculates the number of payment periods.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 108. See also amortTbl(), page 6.

Financial function that calculates the amount of each payment.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 108. See also amortTbl(), page 6.

Financial function that calculates the present value.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 108. See also amortTbl(), page 6.

<table>
<thead>
<tr>
<th>TVM argument*</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>Number of payment periods</td>
<td>real number</td>
</tr>
<tr>
<td>(I)</td>
<td>Annual interest rate</td>
<td>real number</td>
</tr>
<tr>
<td>(PV)</td>
<td>Present value</td>
<td>real number</td>
</tr>
<tr>
<td>(Pmt)</td>
<td>Payment amount</td>
<td>real number</td>
</tr>
<tr>
<td>(FV)</td>
<td>Future value</td>
<td>real number</td>
</tr>
<tr>
<td>(PpY)</td>
<td>Payments per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>(CpY)</td>
<td>Compounding periods per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>(PmtAt)</td>
<td>Payment due at the end or beginning of each period, default=end</td>
<td>integer (0=end, 1=beginning)</td>
</tr>
</tbody>
</table>

* These time-value-of-money argument names are similar to the TVM variable names (such as tvm.pv and tvm.pmt) that are used by the Calculator application’s finance solver. Financial functions, however, do not store their argument values or results to the TVM variables.
TwoVar $X, Y, [Freq], [Category, Include]$

Calculates the TwoVar statistics. A summary of results is stored in the `stat.results` variable. (See page 98.)

All the lists must have equal dimension except for `Include`.

$X$ and $Y$ are lists of independent and dependent variables.

$Freq$ is an optional list of frequency values. Each element in $Freq$ specifies the frequency of occurrence for each corresponding $X$ and $Y$ data point. The default value is 1. All elements must be integers $\geq 0$.

`Category` is a list of numeric category codes for the corresponding $X$ and $Y$ data.

`Include` is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists $X$, $Freq$, or $Category$ results in a void for the corresponding element of all those lists. An empty element in any of the lists $X1$ through $X20$ results in a void for the corresponding element of all those lists. For more information on empty elements, see page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.$\bar{X}$</td>
<td>Mean of $x$ values</td>
</tr>
<tr>
<td>stat.$\Sigma x$</td>
<td>Sum of $x$ values</td>
</tr>
<tr>
<td>stat.$\Sigma x^2$</td>
<td>Sum of $x^2$ values</td>
</tr>
<tr>
<td>stat.$s_x$</td>
<td>Sample standard deviation of $x$</td>
</tr>
<tr>
<td>stat.$\sigma_x$</td>
<td>Population standard deviation of $x$</td>
</tr>
<tr>
<td>stat.$n$</td>
<td>Number of data points</td>
</tr>
<tr>
<td>stat.$\bar{Y}$</td>
<td>Mean of $y$ values</td>
</tr>
<tr>
<td>stat.$\Sigma y$</td>
<td>Sum of $y$ values</td>
</tr>
<tr>
<td>stat.$\Sigma y^2$</td>
<td>Sum of $y^2$ values</td>
</tr>
<tr>
<td>stat.$s_y$</td>
<td>Sample standard deviation of $y$</td>
</tr>
<tr>
<td>stat.$\sigma_y$</td>
<td>Population standard deviation of $y$</td>
</tr>
<tr>
<td>stat.$\Sigma xy$</td>
<td>Sum of $x \cdot y$ values</td>
</tr>
<tr>
<td>stat.$r$</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.$\text{Min}X$</td>
<td>Minimum of $x$ values</td>
</tr>
<tr>
<td>stat.$Q_1X$</td>
<td>1st Quartile of $x$</td>
</tr>
<tr>
<td>stat.$\text{Median}X$</td>
<td>Median of $x$</td>
</tr>
<tr>
<td>stat.$Q_3X$</td>
<td>3rd Quartile of $x$</td>
</tr>
<tr>
<td>stat.$\text{Max}X$</td>
<td>Maximum of $x$ values</td>
</tr>
<tr>
<td>stat.$\text{Min}Y$</td>
<td>Minimum of $y$ values</td>
</tr>
<tr>
<td>stat.$Q_1Y$</td>
<td>1st Quartile of $y$</td>
</tr>
</tbody>
</table>
### Table of Output Variables and Descriptions

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.MedY</code></td>
<td>Median of y</td>
</tr>
<tr>
<td><code>stat.Q3Y</code></td>
<td>3rd Quartile of y</td>
</tr>
<tr>
<td><code>stat.MaxY</code></td>
<td>Maximum of y values</td>
</tr>
<tr>
<td><code>stat.Sxy-x̄ȳ^2</code></td>
<td>Sum of squares of deviations from the mean of x</td>
</tr>
<tr>
<td><code>stat.Sy-ȳ^2</code></td>
<td>Sum of squares of deviations from the mean of y</td>
</tr>
</tbody>
</table>

### Unit Vector (`unitV()`)

`unitV(Vector1) → vector`

Returns either a row- or column-unit vector, depending on the form of `Vector1`. `Vector1` must be either a single-row matrix or a single-column matrix.

#### Examples

- `unitV([[1, 2, 1]]) → \begin{bmatrix} 0.408248 \\ 0.816497 \\ 0.408248 \end{bmatrix}`
- `unitV([[1, 2], [2, 3]]) → \begin{bmatrix} 0.267261 \\ 0.534522 \\ 0.801784 \end{bmatrix}`

### Unlocking Variables (`unLock`)

`unLock Var1 [, Var2] [, Var3] ...`

Unlocks the specified variables or variable group. Locked variables cannot be modified or deleted.

#### Examples

- `a:=65`
- `Lock a`  
  `Done`
- `getLockInfo(a)` → `1`
- `a:=75` → "Error: Variable is locked."
- `DelVar a` → "Error: Variable is locked."
- `Unlock a`  
  `Done`
- `a:=75`
- `DelVar a`  
  `Done`

### Population Variance (`varPop()`)

`varPop(List[, freqList]) → expression`

Returns the population variance of `List`.

- `varPop(\{5,10,15,20,25,30\}) → 72.9167`

Each `freqList` element counts the number of consecutive occurrences of the corresponding element in `List`.

**Note:** `List` must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 132.
varSamp()  

\[ \text{varSamp}([\text{List}_1, \text{freqList}]) \Rightarrow \text{expression} \]

Returns the sample variance of \( \text{List}_1 \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List}_1 \).

**Note:** \( \text{List} \) must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 132.

\[ \text{varSamp}([\text{Matrix}_1, \text{freqMatrix}]) \Rightarrow \text{matrix} \]

Returns a row vector containing the sample variance of each column in \( \text{Matrix}_1 \).

Each \( \text{freqMatrix} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{Matrix}_1 \).

If an element in either matrix is empty (void), that element is ignored, and the corresponding element in the other matrix is also ignored. For more information on empty elements, see page 132.

**Note:** \( \text{Matrix}_1 \) must contain at least two rows.

\\

## warnCodes()

\[ \text{warnCodes}(\text{Expr1}, \text{StatusVar}) \Rightarrow \text{expression} \]

Evaluates expression \( \text{Expr1} \), returns the result, and stores the codes of any generated warnings in the \( \text{StatusVar} \) list variable. If no warnings are generated, this function assigns \( \text{StatusVar} \) an empty list.

\( \text{Expr1} \) can be any valid TI-Nspire™ or TI-Nspire™ CAS math expression. You cannot use a command or assignment as \( \text{Expr1} \).

\( \text{StatusVar} \) must be a valid variable name.

For a list of warning codes and associated messages, see page 144.

\\

## when()

\[ \text{when}(\text{Condition}, \text{trueResult} [, \text{falseResult}], \text{unknownResult}) \Rightarrow \text{expression} \]

Returns \( \text{trueResult} \), \( \text{falseResult} \), or \( \text{unknownResult} \), depending on whether \( \text{Condition} \) is true, false, or unknown. Returns the input if there are too few arguments to specify the appropriate result.

Omit both \( \text{falseResult} \) and \( \text{unknownResult} \) to make an expression defined only in the region where \( \text{Condition} \) is true.

Use an \text{undef} \( \text{falseResult} \) to define an expression that graphs only on an interval.

\text{when()} \ is helpful for defining recursive functions.

\\

\[ \text{when}(x<0,x+3) | x=5 \quad \text{undef} \]

\[ \text{when}(n>0,n\cdot\text{factorial}(n-1),1) \rightarrow \text{factorial}(n) \quad \text{Done} \]

\[ \text{factorial}(3) \quad 6 \]

\[ 3! \quad 6 \]
While

While Condition
Block
EndWhile

Executes the statements in Block as long as Condition is true.

Block can be either a single statement or a sequence of statements separated with the “:” character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \[\text{alt}+e\] instead of \[\text{enter}\] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

\[\text{xor}\]

BooleanExpr1 xor BooleanExpr2 returns Boolean expression
BooleanList1 xor BooleanList2 returns Boolean list
BooleanMatrix1 xor BooleanMatrix2 returns Boolean matrix

Returns true if BooleanExpr1 is true and BooleanExpr2 is false, or vice versa.

Returns false if both arguments are true or if both are false. Returns a simplified Boolean expression if either of the arguments cannot be resolved to true or false.

Note: See or, page 73.

Integer1 xor Integer2 \(\Rightarrow\) integer

Compares two real integers bit-by-bit using an xor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see PBase2, page 12.

Note: See or, page 73.
z

**zInterval**

zInterval σ₅,List[ᵢ,Freqᵢ,CLevel]\]

(Data list input)

zInterval σₓₓn, [CLevel]

(Summary stats input)

Computes a z confidence interval. A summary of results is stored in the stat.results variable. (See page 98.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval for an unknown population mean</td>
</tr>
<tr>
<td>statₓₓ</td>
<td>Sample mean of the data sequence from the normal random distribution</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat sx</td>
<td>Sample standard deviation</td>
</tr>
<tr>
<td>stat.n</td>
<td>Length of the data sequence with sample mean</td>
</tr>
<tr>
<td>stat.σ</td>
<td>Known population standard deviation for data sequence List</td>
</tr>
</tbody>
</table>

**zInterval_1Prop**

zInterval_1Prop xₙ, [CLevel]

Computes a one-proportion z confidence interval. A summary of results is stored in the stat.results variable. (See page 98.)

x is a non-negative integer.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>stat.ᵦ</td>
<td>The calculated proportion of successes</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.n</td>
<td>Number of samples in data sequence</td>
</tr>
</tbody>
</table>
**zinterval_2Prop**

Computes a two-proportion $z$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 98.)

$x_1$ and $x_2$ are non-negative integers.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>stat.\hat{p}</code></td>
<td>The calculated difference between proportions</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>stat.\hat{p}_1</code></td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.\hat{p}_2</code></td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.n1</code></td>
<td>Sample size in data sequence one</td>
</tr>
<tr>
<td><code>stat.n2</code></td>
<td>Sample size in data sequence two</td>
</tr>
</tbody>
</table>

**zinterval_2Samp**

Computes a two-sample $z$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 98.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>\bar{x}_1 - \bar{x}_2</code></td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>\bar{x}_1</code>, <code>\bar{x}_2</code></td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td><code>\sigma_1</code>, <code>\sigma_2</code></td>
<td>Sample standard deviations for <code>List 1</code> and <code>List 2</code></td>
</tr>
<tr>
<td><code>\sigma_{x1}</code>, <code>\sigma_{x2}</code></td>
<td>Sample standard deviations for <code>List 1</code> and <code>List 2</code></td>
</tr>
<tr>
<td><code>stat.n1</code>, <code>stat.n2</code></td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td><code>stat.r1</code>, <code>stat.r2</code></td>
<td>Known population standard deviations for data sequence <code>List 1</code> and <code>List 2</code></td>
</tr>
</tbody>
</table>
**zTest**

**zTest \( \mu, \sigma, \text{List, } \{\text{Freq, Hypoth}\} \)**

(Data list input)

**zTest \( \mu, \sigma, \bar{\text{List}}, \text{Hypoth}\)**

(Summary stats input)

Performs a z test with frequency freqlist. A summary of results is stored in the stat.results variable. (See page 98.)

Test \( H_0: \mu = \mu_0 \), against one of the following:

- For \( H_a: \mu < \mu_0 \), set Hypoth<0
- For \( H_a: \mu \neq \mu_0 \) (default), set Hypoth=0
- For \( H_a: \mu > \mu_0 \), set Hypoth>0

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>((\bar{x} - \mu_0) / (\sigma / \sqrt{n}))</td>
</tr>
<tr>
<td>stat.P Value</td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.( \bar{x} )</td>
<td>Sample mean of the data sequence in List</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of the data sequence. Only returned for Data input.</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

***zTest_1Prop***

**zTest_1Prop \( p_0, x, n, \text{Hypoth} \)**

Computes a one-proportion z test. A summary of results is stored in the stat.results variable. (See page 98.)

\( x \) is a non-negative integer.

Test \( H_0: p = p_0 \) against one of the following:

- For \( H_a: p > p_0 \), set Hypoth>0
- For \( H_a: p \neq p_0 \) (default), set Hypoth=0
- For \( H_a: p < p_0 \), set Hypoth<0

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.p0</td>
<td>Hypothesized population proportion</td>
</tr>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the proportion</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.( \hat{p} )</td>
<td>Estimated sample proportion</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>
### zTest_2Prop

**zTest_2Prop** \( x_1, n_1, x_2, n_2, [Hypoth] \)

Computes a two-proportion \( z \) test. A summary of results is stored in the `stat.results` variable. (See page 98.)

\( x_1 \) and \( x_2 \) are non-negative integers.

Test \( H_0: p_1 = p_2 \), against one of the following:

- For \( H_0: p_1 > p_2 \), set `Hypoth>0`
- For \( H_0: p_1 \neq p_2 \) (default), set `Hypoth=0`
- For \( H_0: p < p_0 \), set `Hypoth<0`

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.z</code></td>
<td>Standard normal value computed for the difference of proportions</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td><code>stat.\hat{p}_1</code></td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.\hat{p}_2</code></td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.\hat{p}</code></td>
<td>Pooled sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.n1, stat.n2</code></td>
<td>Number of samples taken in trials 1 and 2</td>
</tr>
</tbody>
</table>

### zTest_2Samp

**zTest_2Samp** \( \sigma_1, \sigma_2, List1, List2, [Freq1], [Freq2], [Hypoth] \)

(Data list input)

**zTest_2Samp** \( \sigma_1, \sigma_2, \bar{x}_1, n_1, \bar{x}_2, n_2, [Hypoth] \)

(Summary stats input)

Computes a two-sample \( z \) test. A summary of results is stored in the `stat.results` variable. (See page 98.)

Test \( H_0: \mu_1 = \mu_2 \), against one of the following:

- For \( H_0: \mu_1 < \mu_2 \), set `Hypoth<0`
- For \( H_0: \mu_1 \neq \mu_2 \) (default), set `Hypoth=0`
- For \( H_0: \mu_1 > \mu_2 \), `Hypoth>0`

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 132.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.z</code></td>
<td>Standard normal value computed for the difference of means</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td><code>stat.\bar{x}_1, stat.\bar{x}_2</code></td>
<td>Sample means of the data sequences in <code>List1</code> and <code>List2</code></td>
</tr>
<tr>
<td><code>stat.sx1, stat.sx2</code></td>
<td>Sample standard deviations of the data sequences in <code>List1</code> and <code>List2</code></td>
</tr>
<tr>
<td><code>stat.n1, stat.n2</code></td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>
Symbols

**+ (add)**

\[ \text{Value1} + \text{Value2} \Rightarrow \text{value} \]

Returns the sum of the two arguments.

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
<th>( \Rightarrow )</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>4</td>
<td>( \Rightarrow )</td>
<td>60</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>( \Rightarrow )</td>
<td>64</td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>( \Rightarrow )</td>
<td>68</td>
</tr>
<tr>
<td>68</td>
<td>4</td>
<td>( \Rightarrow )</td>
<td>72</td>
</tr>
</tbody>
</table>

\[ \text{List1} + \text{List2} \Rightarrow \text{list} \]

\[ \text{Matrix1} + \text{Matrix2} \Rightarrow \text{matrix} \]

Returns a list (or matrix) containing the sums of corresponding elements in \( \text{List1} \) and \( \text{List2} \) (or \( \text{Matrix1} \) and \( \text{Matrix2} \)).

Dimensions of the arguments must be equal.

\[ \begin{bmatrix} 22, \pi, \frac{\pi}{2} \end{bmatrix} \rightarrow \text{List1} \]

\[ \begin{bmatrix} 10, 5, \frac{\pi}{2} \end{bmatrix} \rightarrow \text{List2} \]

\[ \text{List1} + \text{List2} \Rightarrow \begin{bmatrix} 22, 3.14159, 1.5708 \end{bmatrix} \]

\[ \text{Matrix1} + \text{Matrix2} \Rightarrow \begin{bmatrix} 10, 5, 1.5708 \end{bmatrix} \]

**Note:** Use \( .+ \) (dot plus) to add an expression to each element.

**- (subtract)**

\[ \text{Value1} - \text{Value2} \Rightarrow \text{value} \]

Returns \( \text{Value1} \) minus \( \text{Value2} \).

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
<th>( \Rightarrow )</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>( \Rightarrow )</td>
<td>4</td>
</tr>
<tr>
<td>( \pi )</td>
<td>( \frac{\pi}{6} )</td>
<td>( \Rightarrow )</td>
<td>2.61799</td>
</tr>
</tbody>
</table>

\[ \text{List1} - \text{List2} \Rightarrow \text{list} \]

\[ \text{Matrix1} - \text{Matrix2} \Rightarrow \text{matrix} \]

Subtracts each element in \( \text{List2} \) (or \( \text{Matrix2} \)) from the corresponding element in \( \text{List1} \) (or \( \text{Matrix1} \)), and returns the results.

Dimensions of the arguments must be equal.

\[ \begin{bmatrix} 22, \pi, \frac{\pi}{2} \end{bmatrix} - \begin{bmatrix} 10, 5, \frac{\pi}{2} \end{bmatrix} \Rightarrow \begin{bmatrix} 12, 1.85841, 0 \end{bmatrix} \]

\[ \begin{bmatrix} 3 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 2 \end{bmatrix} \]

\[ \text{Value} - \text{List1} \Rightarrow \text{list} \]

\[ \text{List1} - \text{Value} \Rightarrow \text{list} \]

Subtracts each \( \text{List1} \) element from \( \text{Value} \) or subtracts \( \text{Value} \) from each \( \text{List1} \) element, and returns a list of the results.

\[ 15 - \{ 10, 15, 20 \} \Rightarrow \{ 5, 0, 5 \} \]

\[ \{ 10, 15, 20 \} - 15 \Rightarrow \{ -5, 0, 5 \} \]
- (subtract)  \( \rightarrow \text{key} \)

\[
\begin{array}{c}
\text{Value} - \text{Matrix1} \Rightarrow \text{matrix} \\
\text{Matrix1} - \text{Value} \Rightarrow \text{matrix}
\end{array}
\]

\[
\begin{array}{ccc}
20 & 1 & 2 \\
3 & 4 & \end{array}
\]

\[
\begin{array}{ccc}
19 & -2 \\
-3 & 16 & \end{array}
\]

\text{Value} - \text{Matrix1} \text{ returns a matrix of Value times the identity matrix minus Matrix1. Matrix1 must be square.}

\text{Matrix1} - \text{Value} \text{ returns a matrix of Value times the identity matrix subtracted from Matrix1. Matrix1 must be square.}

**Note:** Use \( \cdot \) (dot minus) to subtract an expression from each element.

\cdot (multiply)  \( \times \text{key} \)

\[
\begin{array}{c}
\text{Value1} \cdot \text{Value2} \Rightarrow \text{value} \\
\text{List1} \cdot \text{List2} \Rightarrow \text{list}
\end{array}
\]

\[
\begin{array}{c}
2 \cdot 3.45 \\
\{1,2,3\} \cdot \{4,5,6\} \\
\{4,10,18\}
\end{array}
\]

\[
\begin{array}{c}
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix} \times \begin{bmatrix}
7 & 8 \\
7 & 8
\end{bmatrix} \\
\begin{bmatrix}
42 & 48 \\
105 & 120
\end{bmatrix}
\end{array}
\]

\text{Value} \cdot \text{List1} \Rightarrow \text{list}

\text{List1} \cdot \text{Value} \Rightarrow \text{list}

\text{Returns a list containing the products of Value and each element in List1.}

\text{Value} \cdot \text{Matrix1} \Rightarrow \text{matrix}

\text{Matrix1} \cdot \text{Value} \Rightarrow \text{matrix}

\text{Returns a matrix containing the products of Value and each element in Matrix1.}

**Note:** Use \( \cdot \) (dot multiply) to multiply an expression by each element.

/ (divide)  \( \div \text{key} \)

\[
\begin{array}{c}
\text{Value1} / \text{Value2} \Rightarrow \text{value} \\
\text{List1} / \text{List2} \Rightarrow \text{list}
\end{array}
\]

\[
\begin{array}{c}
\frac{2}{3.45} \\
\{1,2,3\} / \{4,5,6\} \\
\{0.25, \frac{2}{5}, \frac{1}{2}\}
\end{array}
\]

\[
\begin{array}{c}
\frac{6}{3.6, \sqrt{6}} \\
\{2,1,2.44949\}
\end{array}
\]

\[
\begin{array}{c}
\frac{7.9,2}{7.9,2} \\
\{1,1,1\} \\
\{18, 14, \sqrt{63}\}
\end{array}
\]

\text{Note:** See also Fraction template, page 1.}

\text{Returns the quotient of Value1 divided by Value2.}

\text{Returns a list containing the quotients of List1 divided by List2.}

\text{Dimensions of the lists must be equal.}

\text{Value} / \text{List1} \Rightarrow \text{list}

\text{List1} / \text{Value} \Rightarrow \text{list}

\text{Returns a list containing the quotients of Value divided by List1 or List1 divided by Value.}
**/ (divide)**

Value / Matrix1 \(\rightarrow\) matrix

Matrix1 / Value \(\rightarrow\) matrix

Returns a matrix containing the quotients of Matrix1 / Value.

**Note:** Use \(\cdot /\) (dot divide) to divide an expression by each element.

<table>
<thead>
<tr>
<th>7 9 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7·9·2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 14 63</td>
</tr>
</tbody>
</table>

**^ (power)**

Value1 \(^\) Value2 \(\Rightarrow\) value

List1 \(^\) List2 \(\Rightarrow\) list

Returns the first argument raised to the power of the second argument.

**Note:** See also Exponent template, page 1.

For a list, returns the elements in List1 raised to the power of the corresponding elements in List2.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

Value \(^\) List1 \(\Rightarrow\) list

Returns Value raised to the power of the elements in List1.

List1 \(^\) Value \(\Rightarrow\) list

Returns the elements in List1 raised to the power of Value.

<table>
<thead>
<tr>
<th>4^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>{2,4,6} {1,2,3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,16,216}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\pi) {1,2,-3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3.14159,9.8696,0.032252}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>{(1,2,3)}^{-2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,\frac{1}{4},\frac{1}{16}}</td>
</tr>
</tbody>
</table>

| \[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^2
\] |
|-----------------|
| \[
\begin{bmatrix}
7 & 10 \\
15 & 22
\end{bmatrix}
\] |

| \[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^{-1}
\] |
|-----------------|
| \[
\begin{bmatrix}
1 & -1 \\
3 & 2
\end{bmatrix}
\] |

| \[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^{-2}
\] |
|-----------------|
| \[
\begin{bmatrix}
11 & -5 \\
2 & 2 \\
15 & 7 \\
4 & 4
\end{bmatrix}
\] |

**x² (square)**

Value\(^2\) \(\Rightarrow\) value

Returns the square of the argument.

List\(^2\) \(\Rightarrow\) list

Returns a list containing the squares of the elements in List1.

squareMatrix\(^2\) \(\Rightarrow\) matrix

Returns the matrix square of squareMatrix1. This is not the same as calculating the square of each element. Use \(^\cdot 2\) to calculate the square of each element.
.+$ \quad (dot \ add) \quad \text{.} + \quad \text{keys}$

Matrix 1 .+$ Matrix 2 \Rightarrow \text{matrix}$
Value .+$ Matrix 1 \Rightarrow \text{matrix}$

Matrix 1 .+$ Matrix 2$ returns a matrix that is the sum of each pair of corresponding elements in Matrix 1 and Matrix 2.

Value .+$ Matrix 1$ returns a matrix that is the sum of Value and each element in Matrix 1.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\begin{bmatrix}
10 & 30 \\
20 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
11 & 32 \\
23 & 44 \\
\end{bmatrix}
\]

\[
5 .+ \begin{bmatrix}
10 & 30 \\
20 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
15 & 35 \\
25 & 45 \\
\end{bmatrix}
\]

$\cdot-$ \quad (dot \ subtract) \quad \text{.} -$ \quad \text{keys}$

Matrix 1 $\cdot-$ Matrix 2 \Rightarrow \text{matrix}
Value $\cdot-$ Matrix 1 \Rightarrow \text{matrix}$

Matrix 1 $\cdot-$ Matrix 2 returns a matrix that is the difference between each pair of corresponding elements in Matrix 1 and Matrix 2.

Value $\cdot-$ Matrix 1 returns a matrix that is the difference of Value and each element in Matrix 1.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
\text{---} & \text{---} \\
\text{---} & \text{---} \\
\end{bmatrix}
\]

\[
5 \cdot- \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
5 & \text{---} \\
\text{---} & \text{---} \\
\end{bmatrix}
\]

$\cdot\cdot$ \quad (dot \ multiply) \quad \text{.} \times \quad \text{keys}$

Matrix 1 $\cdot\cdot$ Matrix 2 \Rightarrow \text{matrix}$
Value $\cdot\cdot$ Matrix 1 \Rightarrow \text{matrix}$

Matrix 1 $\cdot\cdot$ Matrix 2 returns a matrix that is the product of each pair of corresponding elements in Matrix 1 and Matrix 2.

Value $\cdot\cdot$ Matrix 1 returns a matrix containing the products of Value and each element in Matrix 1.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
10 & 40 \\
90 & 160 \\
\end{bmatrix}
\]

\[
5 \cdot\cdot \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
50 & 100 \\
150 & 200 \\
\end{bmatrix}
\]

$\div$ \quad (dot \ divide) \quad \text{.} \div \quad \text{keys}$

Matrix 1 $\div$ Matrix 2 \Rightarrow \text{matrix}$
Value $\div$ Matrix 1 \Rightarrow \text{matrix}$

Matrix 1 $\div$ Matrix 2 returns a matrix that is the quotient of each pair of corresponding elements in Matrix 1 and Matrix 2.

Value $\div$ Matrix 1 returns a matrix that is the quotient of Value and each element in Matrix 1.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\div \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
\frac{1}{10} & \frac{1}{10} \\
\frac{1}{10} & \frac{1}{10} \\
\end{bmatrix}
\]

\[
5 \div \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
\frac{1}{2} & \frac{1}{4} \\
\frac{1}{6} & \frac{1}{8} \\
\end{bmatrix}
\]

$\cdot^\wedge$ \quad (dot \ power) \quad \text{.} ^\wedge \quad \text{keys}$

Matrix 1 $\cdot^\wedge$ Matrix 2 \Rightarrow \text{matrix}$
Value $\cdot^\wedge$ Matrix 1 \Rightarrow \text{matrix}$

Matrix 1 $\cdot^\wedge$ Matrix 2 returns a matrix where each element in Matrix 2 is the exponent for the corresponding element in Matrix 1.

Value $\cdot^\wedge$ Matrix 1 returns a matrix where each element in Matrix 1 is the exponent for Value.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\cdot^\wedge \begin{bmatrix}
0 & 2 \\
3 & -1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 4 \\
27 & \frac{1}{4} \\
\end{bmatrix}
\]

\[
5 \cdot^\wedge \begin{bmatrix}
0 & 2 \\
3 & -1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & 25 \\
125 & \frac{1}{5} \\
\end{bmatrix}
\]
-(negate)

```
-\text{Value1} \Rightarrow \text{value}
-\text{List1} \Rightarrow \text{list}
-\text{Matrix1} \Rightarrow \text{matrix}
```

Returns the negation of the argument.

For a list or matrix, returns all the elements negated.

If the argument is a binary or hexadecimal integer, the negation gives the two’s complement.

```
-2.43
\{-1.0,4.12E19\}
\{1.0,-0.4,12E19\}
```

In Bin base mode:

```
\text{Important: Zero, not the letter O}
```

```
0b100101 \Rightarrow \text{Dec} 37
-0b100101
0b1111111111111111111111111111111
\text{Ans} \Rightarrow \text{Dec} -37
```

To see the entire result, press \( \triangleright \), and then use \( \downarrow \) and \( \uparrow \) to move the cursor.

% (percent)

```
\text{Value1} \% \Rightarrow \text{value}
\text{List1} \% \Rightarrow \text{list}
\text{Matrix1} \% \Rightarrow \text{matrix}
```

Returns \text{argument} \( \frac{\text{argument}}{100} \)

For a list or matrix, returns a list or matrix with each element divided by 100.

```
13\% 0.13
\{1,10,100\}\% \{0.01,0.1,1.\}
```

Press Ctrl+Enter to evaluate:
\( \text{=} \) (equal)

\[ \text{Expr1} = \text{Expr2} \Rightarrow \text{Boolean expression} \]
\[ \text{List1} = \text{List2} \Rightarrow \text{Boolean list} \]
\[ \text{Matrix1} = \text{Matrix2} \Rightarrow \text{Boolean matrix} \]

Returns true if \( \text{Expr1} \) is determined to be equal to \( \text{Expr2} \).
Returns false if \( \text{Expr1} \) is determined to not be equal to \( \text{Expr2} \).
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \( \rightarrow \) instead of \( \text{enter} \) at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Example function that uses math test symbols: \( =, \neq, <, >, \geq \)

Define \( g(x) \)=Func
If \( x \leq -5 \) Then
Return 5
ElseIf \( x > -5 \) and \( x < 0 \) Then
Return \( -x \)
ElseIf \( x \geq 0 \) and \( x \neq 10 \) Then
Return \( x \)
ElseIf \( x = 10 \) Then
Return 3
EndIf
EndFunc

Result of graphing \( g(x) \)

\( \neq \) (not equal)

\[ \text{Expr1} \neq \text{Expr2} \Rightarrow \text{Boolean expression} \]
\[ \text{List1} \neq \text{List2} \Rightarrow \text{Boolean list} \]
\[ \text{Matrix1} \neq \text{Matrix2} \Rightarrow \text{Boolean matrix} \]

Returns true if \( \text{Expr1} \) is determined to be not equal to \( \text{Expr2} \).
Returns false if \( \text{Expr1} \) is determined to be equal to \( \text{Expr2} \).
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing \( /\) =

\(<\) (less than)

\[ \text{Expr1} < \text{Expr2} \Rightarrow \text{Boolean expression} \]
\[ \text{List1} < \text{List2} \Rightarrow \text{Boolean list} \]
\[ \text{Matrix1} < \text{Matrix2} \Rightarrow \text{Boolean matrix} \]

Returns true if \( \text{Expr1} \) is determined to be less than \( \text{Expr2} \).
Returns false if \( \text{Expr1} \) is determined to be greater than or equal to \( \text{Expr2} \).
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.
\(<\) (less or equal)

$\text{Expr1} \leq \text{Expr2} \Rightarrow \text{Boolean expression}

\text{List1} \leq \text{List2} \Rightarrow \text{Boolean list}

\text{Matrix1} \leq \text{Matrix2} \Rightarrow \text{Boolean matrix}

Returns true if $\text{Expr1}$ is determined to be less than or equal to $\text{Expr2}$.

Returns false if $\text{Expr1}$ is determined to be greater than $\text{Expr2}$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

\textbf{Note:} You can insert this operator from the keyboard by typing $\leq$

\(\geq\) (greater or equal)

$\text{Expr1} \geq \text{Expr2} \Rightarrow \text{Boolean expression}

\text{List1} \geq \text{List2} \Rightarrow \text{Boolean list}

\text{Matrix1} \geq \text{Matrix2} \Rightarrow \text{Boolean matrix}

Returns true if $\text{Expr1}$ is determined to be greater than or equal to $\text{Expr2}$.

Returns false if $\text{Expr1}$ is determined to be less than $\text{Expr2}$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

\textbf{Note:} You can insert this operator from the keyboard by typing $\geq$

\(\Rightarrow\) (logical implication)

BooleanExpression1 $\Rightarrow$ BooleanExpression2 returns Boolean expression

BooleanList1 $\Rightarrow$ BooleanList2 returns Boolean list

BooleanMatrix1 $\Rightarrow$ BooleanMatrix2 returns Boolean matrix

Integer1 $\Rightarrow$ Integer2 returns Integer

Evaluates the expression not $<$argument1$>$ or $<$argument2$>$ and returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

\textbf{Note:} You can insert this operator from the keyboard by typing $\Rightarrow$
\( \iff (\text{logical double implication, XNOR}) \)\n
\[ \text{BooleanExpr1} \iff \text{BooleanExpr2} \Rightarrow \text{Boolean expression} \]

\[ \text{BooleanList1} \iff \text{BooleanList2} \Rightarrow \text{Boolean list} \]

\[ \text{BooleanMatrix1} \iff \text{BooleanMatrix2} \Rightarrow \text{Boolean matrix} \]

\[ \text{Integer1} \iff \text{Integer2} \Rightarrow \text{Integer} \]

Returns the negation of an XOR Boolean operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing \( \iff \).

\[ 5 \times 3 \text{ xor } 3 > 5 \]

\[ 3 \times 3 \iff 3 > 5 \]

\[ \{1, 2, 3\} \text{ xor } \{3, 2, 1\} \]

\[ \{1, 2, 3\} \iff \{3, 2, 1\} \]

\( ! (\text{factorial}) \)

\[ \text{Value1} \Rightarrow \text{value} \]

\[ \text{List1} \Rightarrow \text{list} \]

\[ \text{Matrix1} \Rightarrow \text{matrix} \]

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

\( & (\text{append}) \)

\[ \text{String1} \& \text{String2} \Rightarrow \text{string} \]

Returns a text string that is String2 appended to String1.

\( d() (\text{derivative}) \)

\[ d(\text{Expr1}, \text{Var}, \text{Order}) \mid \text{Var}=\text{Value} \Rightarrow \text{value} \]

\[ d(\text{List1}, \text{Var}, \text{Order}) \Rightarrow \text{list} \]

\[ d(\text{Matrix1}, \text{Var}, \text{Order}) \Rightarrow \text{matrix} \]

Except when using the first syntax, you must store a numeric value in variable Var before evaluating d(). Refer to the examples.

\( d() \) can be used for calculating first and second order derivative at a point numerically, using auto differentiation methods.

**Order:** If included, must be 1 or 2. The default is 1.

**Note:** You can insert this function from the keyboard by typing \( \text{derivative(}...\).\)

**Note:** See also First derivative, page 5 or Second derivative, page 5.

**Note:** The \( d() \) algorithm has a limitation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of \( x \cdot (x^2+x)^{1/3} \) at \( x=0 \) is equal to 0. However, because the first derivative of the subexpression \( (x^2+x)^{1/3} \) is undefined at \( x=0 \), and this value is used to calculate the derivative of the total expression, \( d() \) reports the result as undefined and displays a warning message.

If you encounter this limitation, verify the solution graphically. You can also try using centralDiff().

\[ \frac{d}{dx} \left( |x| \right) \bigg|_{x=0} \]

\[ x=0: \frac{d}{dx} \left( |x| \right) \]

\[ x=3: \frac{d}{dx} \left( \left( x^2, x^3, x^4 \right) \right) \]

\[ \frac{d}{dx} \left( x \cdot (x^2+x)^{1/3} \right) \bigg|_{x=0} \]

\[ \text{centralDiff} \left( x \cdot (x^2+x)^{1/3}, x \bigg|_{x=0} \right) \]

\[ 0.000033 \]
\[ \int (Expr1, Var, Lower, Upper) \Rightarrow \text{value} \]

Returns the integral of \( Expr1 \) with respect to the variable \( Var \) from \( Lower \) to \( Upper \). Can be used to calculate the definite integral numerically, using the same method as \( \text{nInt()} \).

**Note:** You can insert this function from the keyboard by typing \( \text{integral(} \ldots \text{)} \).

**Note:** See also \( \text{nInt()} \), page 68, and **Definite integral template**, page 5.

\[ \sqrt{()} (\text{square root}) \]

\[ \sqrt{(Value)} \Rightarrow \text{value} \]
\[ \sqrt{(List)} \Rightarrow \text{list} \]

Returns the square root of the argument.

For a list, returns the square roots of all the elements in \( \text{List1} \).

**Note:** You can insert this function from the keyboard by typing \( \text{sqrt(} \ldots \text{)} \).

**Note:** See also **Square root template**, page 1.

\[ \Pi() (\text{prodSeq}) \]

\[ \Pi(Expr1, Var, Low, High) \Rightarrow \text{expression} \]

**Note:** You can insert this function from the keyboard by typing \( \text{prodSeq(} \ldots \text{)} \).

Evaluates \( Expr1 \) for each value of \( Var \) from \( Low \) to \( High \), and returns the product of the results.

**Note:** See also **Product template (\( \Pi) \)**, page 4.

\( \Pi(Expr1, Var, Low, Low-1) \Rightarrow 1 \)

\( \Pi(Expr1, Var, Low, High) \Rightarrow 1/\Pi(Expr1, Var, High+1, Low-1) \) if \( High < Low-1 \)

The product formulas used are derived from the following reference:

\( \Sigma \) (sumSeq)  

\[ \Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \text{expression} \]

**Note:** You can insert this function from the keyboard by typing `sumSeq(...)`. Evaluates Expr1 for each value of Var from Low to High, and returns the sum of the results.

**Note:** See also **Sum template**, page 4.

\[ \Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 0 \]

\[ \Sigma(\text{Expr1}, \text{Var}, \text{High}, \text{Low}) \Rightarrow \Sigma(\text{Expr1}, \text{Var}, \text{High}+1, \text{Low}-1) \text{ if } \text{High} < \text{Low}-1 \]

The summation formulas used are derived from the following reference:


**\( \Sigma \) Int()**

\[ \Sigma \text{Int}(\text{NPmt1}, \text{NPmt2}, \text{N}, \text{I}, \text{PV}, \{\text{Pmt}\}, \{\text{FV}\}, \{\text{PpY}\}, \{\text{CpY}\}, \{\text{PmtAt}\}, \{\text{roundValue}\}) \Rightarrow \text{value} \]

\[ \Sigma \text{Int}(\text{NPmt1}, \text{NPmt2}, \text{amortTable}) \Rightarrow \text{value} \]

Amortization function that calculates the sum of the interest during a specified range of payments.

\( \text{NPmt1} \) and \( \text{NPmt2} \) define the start and end boundaries of the payment range.

\( \text{N}, \text{I}, \text{PV}, \text{Pmt}, \text{FV}, \text{PpY}, \text{CpY}, \text{and PmtAt} \) are described in the table of TVM arguments, page 108.

- If you omit \( \text{Pmt} \), it defaults to \( \text{Pmt} \equiv \text{tvmPmt}(\text{N}, \text{I}, \text{PV}, \text{FV}, \text{PpY}, \text{CpY}) \).
- If you omit \( \text{FV} \), it defaults to \( \text{FV} \equiv 0 \).
- The defaults for \( \text{PpY}, \text{CpY}, \text{and PmtAt} \) are the same as for the TVM functions.

\( \text{roundValue} \) specifies the number of decimal places for rounding. Default=2.

\( \Sigma \text{Int}(\text{NPmt1}, \text{NPmt2}, \text{amortTable}) \) calculates the sum of the interest based on amortization table \( \text{amortTable} \). The \( \text{amortTable} \) argument must be a matrix in the form described under \( \text{amortTbl()} \), page 6.

**Note:** See also **\( \Sigma \) Prn()**, below, and **\( \Sigma \) Bal()**, page 12.

\[ \Sigma \text{Int}(1,3,12,4.75,20000,12,12) \]

\[ \Sigma \text{Int}(1,3,\text{tbl}) \]

\[ \text{tbl} \equiv \text{amortTbl}(12,12,4.75,20000,12,12) \]
Amortization function that calculates the sum of the principal during a specified range of payments. NPmt1 and NPmt2 define the start and end boundaries of the payment range. 

- If you omit \( Pmt \), it defaults to \( Pmt = \text{tvmPmt}(N, I, PV, FV, PpY, CpY, PmtAt) \).
- If you omit \( FV \), it defaults to \( FV = 0 \).
- The defaults for \( PpY \), \( CpY \), and \( PmtAt \) are the same as for the TVM functions.

\( roundValue \) specifies the number of decimal places for rounding. Default=2.

\( \Sigma \text{Prn}(\text{NPmt1, NPmt2, amortTable}) \) calculates the sum of the principal paid based on amortization table \( \text{amortTable} \). The \( \text{amortTable} \) argument must be a matrix in the form described under \( \text{amortTbl}() \), page 6.

Note: See also \( \Sigma \text{Int}() \), above, and \( \text{Bal}() \), page 12.

### # (indirection)

\# \text{varNameString}

Refers to the variable whose name is \text{varNameString}. This lets you use strings to create variable names from within a function.

\text{xyz}:=12

\#("x" & "y" & "z")

12

Creates or refers to the variable \text{xyz}.

10 \rightarrow r

"r" \rightarrow \text{s1}

"r"

10

Returns the value of the variable \( r \) whose name is stored in variable \text{s1}.

### E (scientific notation)

\text{mantissa}\times10^{\text{exponent}}

Enters a number in scientific notation. The number is interpreted as \text{mantissa} \times 10^{\text{exponent}}.

Hint: If you want to enter a power of 10 without causing a decimal value result, use \text{10}^{\text{integer}}.

Note: You can insert this operator from the computer keyboard by typing \text{@E}. For example, type \text{2.3@E4} to enter \text{2.3E4}.
This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies \( \text{Expr1} \) by \( \pi/200 \).

In Degree angle mode, multiplies \( \text{Expr1} \) by \( \pi/100 \).

In Gradian mode, returns \( \text{Expr1} \) unchanged.

Note: You can insert this symbol from the computer keyboard by typing \( \text{@g} \).

This function gives you a way to specify a radian angle while in Degree or Gradian mode.

In Degree angle mode, multiplies the argument by \( 180/\pi \).

In Radian angle mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by \( 200/\pi \).

Hint: Use \( \text{R} \) if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

Note: You can insert this symbol from the computer keyboard by typing \( \text{@r} \).

This function gives you a way to specify a degree angle while in Gradian or Radian mode.

In Radian angle mode, multiplies the argument by \( \pi/180 \).

In Degree angle mode, returns the argument unchanged.

In Gradian angle mode, multiplies the argument by \( 10/9 \).

Note: You can insert this symbol from the computer keyboard by typing \( \text{@d} \).

Returns \( dd+\frac{(mm/60)+(ss.ss/3600)}{} \).

This base-60 entry format lets you:

- Enter an angle in degrees/minutes/seconds without regard to the current angle mode.
- Enter time as hours/minutes/seconds.

Note: Follow \( ss.ss \) with two apostrophes ("), not a quote symbol (").
\( \angle (\text{angle}) \)

\[ [\text{Radius}, \angle \text{Angle}] \Rightarrow \text{vector} \]
(polar input)

\[ [\text{Radius}, \angle \text{Angle}, Z\text{Coordinate}] \Rightarrow \text{vector} \]
(cylindrical input)

\[ [\text{Radius}, \angle \text{Angle}, \angle \text{Angle}] \Rightarrow \text{vector} \]
(spherical input)

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

**Note:** You can insert this symbol from the computer keyboard by typing \( @< \).

\((\text{Magnitude } \angle \text{Angle}) \Rightarrow \text{complexValue} \)
(polar input)

Enters a complex value in \((r \angle \theta)\) polar form. The \(\angle\) is interpreted according to the current Angle mode setting.

\(_{\_}\) (underscore as an empty element)

See "Empty (Void) Elements", page 132.

\(10^()\)

\(10^ \text{ (Value1)} \Rightarrow \text{value}\)

\(10^ \text{ (List1)} \Rightarrow \text{list}\)

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in \(\text{List1}\).

\(10^\text{ (squareMatrix1)} \Rightarrow \text{squareMatrix}\)

Returns 10 raised to the power of \(\text{squareMatrix1}\). This is not the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

\(^{-1}\) (reciprocal)

\(\text{Value1}^{-1} \Rightarrow \text{value}\)

\(\text{List1}^{-1} \Rightarrow \text{list}\)

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in \(\text{List1}\).

\(\text{squareMatrix1}^{-1} \Rightarrow \text{squareMatrix}\)

Returns the inverse of \(\text{squareMatrix1}\).

\(\text{squareMatrix1}\) must be a non-singular square matrix.
The constraint ("|") symbol serves as a binary operator. The operand to the left of | is an expression. The operand to the right of | specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after | must be joined by logical "and" or "or" operators.

The constraint operator provides three basic types of functionality:

- Substitutions
- Interval constraints
- Exclusions

Substitutions are in the form of an equality, such as $x=3$ or $y=\sin(x)$. To be most effective, the left side should be a simple variable. $Expr|Variable=value$ will substitute $value$ for every occurrence of $Variable$ in $Expr$.

Interval constraints take the form of one or more inequalities joined by logical "and" or "or" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

Exclusions use the "not equals" ($\neq$ or /=) relational operator to exclude a specific value from consideration.

---

### (store)

<table>
<thead>
<tr>
<th>$\leftarrow$ (store)</th>
<th>ctrl var key</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Value \leftarrow Var$</td>
<td></td>
</tr>
<tr>
<td>$List \leftarrow Var$</td>
<td></td>
</tr>
<tr>
<td>$Matrix \leftarrow Var$</td>
<td></td>
</tr>
<tr>
<td>$Expr \leftarrow Function(Param1,...)$</td>
<td></td>
</tr>
<tr>
<td>$List \leftarrow Function(Param1,...)$</td>
<td></td>
</tr>
<tr>
<td>$Matrix \leftarrow Function(Param1,...)$</td>
<td></td>
</tr>
</tbody>
</table>

If the variable $Var$ does not exist, creates it and initializes it to $Value$, $List$, or $Matrix$.

If the variable $Var$ already exists and is not locked or protected, replaces its contents with $Value$, $List$, or $Matrix$.

**Note:** You can insert this operator from the keyboard by typing $\leftarrow$ as a shortcut. For example, type $\pi/4 \leftarrow myvar$. 

---

### Example

$x+1|\ x=3$  
$x+55|\ x=\sin(55)$

### Graphs

- **f1(x)**: $x^2$, $x \leq 1$ or $x \geq 2$
- **f2(x)**: $x^2$, $x > 1$ and $x < 2$

---

### Values

| $\pi/4 \rightarrow myvar$ | 0.785398 |
| $2\cdot\cos(x) \rightarrow y_1(x)$ | Done |
| $\{1,2,3,4\} \rightarrow lst5$ | $\{1,2,3,4\}$ |
| $\{1\ 2\ 3\ 4\ 5\ 6\} \rightarrow matg$ | $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ |
| "Hello" $\rightarrow str1$ | "Hello" |
### := (assign)

- **Var := Value**
- **Var := List**
- **Var := Matrix**
- **Function(Param1, ... ) := Expr**
- **Function(Param1, ... ) := List**
- **Function(Param1, ... ) := Matrix**

If variable `Var` does not exist, creates `Var` and initializes it to `Value`, `List`, or `Matrix`.

If `Var` already exists and is not locked or protected, replaces its contents with `Value`, `List`, or `Matrix`.

### © (comment)

- **© [text]**

  © processes text as a comment line, allowing you to annotate functions and programs that you create.

  © can be at the beginning or anywhere in the line. Everything to the right of ©, to the end of the line, is the comment.

  **Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `·` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

### 0b, 0h

**0b** binaryNumber  
**0h** hexadecimalNumber

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

---

```
myvar := \frac{\pi}{4}  \quad .785398
```

```
vF(x) := -2 \cdot \cos(x)  \quad \text{Done}
```

```
lst5 := \{1,2,3,4\}  \quad \{1,2,3,4\}
```

```
matG := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}  \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
```

```
str1 := "Hello"  \quad "Hello"
```

```
Define g(n) = Func  
© Declare variables  
Local i, result
result := 0
For i,1,n,1 © Loop n times
result := result + i^2
EndFor  
Return result
EndFunc  \quad \text{Done}
```

```
g(3)  \quad 14
```

```
In Dec base mode:
0b10 + 0hF + 10  \quad 27
```

```
In Bin base mode:
0b10 + 0hF + 10  \quad 0b11011
```

```
In Hex base mode:
0b10 + 0hF + 10  \quad 0h1B
```
Empty (Void) Elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ Software allows empty, or void, data elements so you can proceed with the nearly complete data rather than having to start over or discard the incomplete cases.

You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “Graphing spreadsheet data.”

The delVoid() function lets you remove empty elements from a list. The isVoid() function lets you test for an empty element. For details, see delVoid(), page 29, and isVoid(), page 49.

Note: To enter an empty element manually in a math expression, type “_” or the keyword void. The keyword void is automatically converted to a “_” symbol when the expression is evaluated. To type “_” on the handheld, press [ctrl] [w].

Calculations involving void elements

The majority of calculations involving a void input will produce a void result. See special cases below.

\[
\begin{align*}
  \text{gcd}(100,_) &= _ \\
  3+ &= _ \\
  \{5,_,10\} &- \{3,6,9\} = \{2,_,1\}
\end{align*}
\]

List arguments containing void elements

The following functions and commands ignore (skip) void elements found in list arguments.

- count, countIf, cumulativeSum, freqTableList, frequency, max, mean, median, product, stDevPop, stDevSamp, sum, sumIf, varPop, and varSamp, as well as regression calculations, OneVar, TwoVar, and FiveNumSummary statistics, confidence intervals, and stat tests

\[
\begin{align*}
  \text{sum}\{2,_,3,5,6,6\} &= 16.6 \\
  \text{median}\{1,2,_,_,3\} &= 2 \\
  \text{cumulativeSum}\{1,2,_,4,5\} &= \{1,3,_,7,12\} \\
  \text{cumulativeSum}\begin{bmatrix} 1 & 2 \\ 3 & _ \\ 5 & 6 \end{bmatrix} &= \begin{bmatrix} 1 & 2 \\ 4 & _ \\ 9 & 8 \end{bmatrix}
\end{align*}
\]

SortA and SortD move all void elements within the first argument to the bottom.

\[
\begin{align*}
  \{5,4,3,_,1\} &\rightarrow \text{list1} & \{5,4,3,_,1\} \\
  \{5,4,3,2,1\} &\rightarrow \text{list2} & \{5,4,3,2,1\}
\end{align*}
\]

SortA.list1.list2 Done

\[
\begin{align*}
  \text{list1} &= \{1,3,4,5,\_\} \\
  \text{list2} &= \{1,3,4,5,2\}
\end{align*}
\]

\[
\begin{align*}
  \{1,2,3,_,5\} &\rightarrow \text{list1} & \{1,2,3,_,5\} \\
  \{1,2,3,4,5\} &\rightarrow \text{list2} & \{1,2,3,4,5\}
\end{align*}
\]

SortD.list1.list2 Done

\[
\begin{align*}
  \text{list1} &= \{5,3,2,1,\_\} \\
  \text{list2} &= \{5,3,2,1,4\}
\end{align*}
\]
In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

\[ H:=\{1,2,3,4,5\} \quad I:=\{2,3,5,6,6\} \]
\[ \text{LinRegMx } H,I \quad \text{Done} \]
\[ \text{stat.Resid} \]
\[ \{0.434286, -0.862857, 0.011429, 0.44\} \]
\[ \text{stat.XReg} \]
\[ \{1, 2, 3, 4, 5\} \]
\[ \text{stat.YReg} \]
\[ \{2, 3, 5, 6, 6\} \]
\[ \text{stat.FreqReg} \]
\[ \{1, 1, 1, 1, 1\} \]

An omitted category in regressions introduces a void for the corresponding element of the residual.

\[ H:=\{1,3,4,5\} \quad I:=\{2,3,5,6,6\} \]
\[ \text{cat}:=\{"M", "M", "F", "F"\} \quad \text{incl}:=\{"F"\} \]
\[ \{"F"\} \]
\[ \text{LinRegMx } H,I,1,\text{cat,incl} \quad \text{Done} \]
\[ \text{stat.Resid} \]
\[ \{1, 0, 0, 0\} \]
\[ \text{stat.XReg} \]
\[ \{1, 4, 5\} \]
\[ \text{stat.YReg} \]
\[ \{1, 5, 6, 6\} \]
\[ \text{stat.FreqReg} \]
\[ \{1, 1, 1\} \]

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

\[ H:=\{1,3,4,5\} \quad I:=\{2,3,5,6,6\} \]
\[ \text{LinRegMx } H,I,2,\{1,0,1,1\} \quad \text{Done} \]
\[ \text{stat.Resid} \]
\[ \{0.069231, -0.276923, 0.207692\} \]
\[ \text{stat.XReg} \]
\[ \{1, 4, 5\} \]
\[ \text{stat.YReg} \]
\[ \{2, 5, 6, 6\} \]
\[ \text{stat.FreqReg} \]
\[ \{1, 1, 1\} \]
Shortcuts for Entering Math Expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression \( \sqrt{6} \), you can type \( \text{sqrt}(6) \) on the entry line. When you press [enter], the expression \( \text{sqrt}(6) \) is changed to \( \sqrt{6} \). Some shortcuts are useful from both the handheld and the computer keyboard. Others are useful primarily from the computer keyboard.

**From the Handheld or Computer Keyboard**

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi )</td>
<td>( \text{pi} )</td>
</tr>
<tr>
<td>( \theta )</td>
<td>( \text{theta} )</td>
</tr>
<tr>
<td>( \infty )</td>
<td>( \text{infinity} )</td>
</tr>
<tr>
<td>( \leq )</td>
<td>( \leq )</td>
</tr>
<tr>
<td>( \geq )</td>
<td>( \geq )</td>
</tr>
<tr>
<td>( \neq )</td>
<td>( \neq )</td>
</tr>
<tr>
<td>( \Rightarrow ) (logical implication)</td>
<td>( \Rightarrow )</td>
</tr>
<tr>
<td>( \Leftrightarrow ) (logical double implication, XNOR)</td>
<td>( \Leftrightarrow )</td>
</tr>
<tr>
<td>( \rightarrow ) (store operator)</td>
<td>( \rightarrow )</td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>( \sqrt{0} )</td>
<td>( \text{sqrt}(...) )</td>
</tr>
<tr>
<td>( \Sigma() ) (Sum template)</td>
<td>( \text{sumSeq}(...) )</td>
</tr>
<tr>
<td>( \Pi() ) (Product template)</td>
<td>( \text{prodSeq}(...) )</td>
</tr>
<tr>
<td>( \sin(), \cos(), \ldots )</td>
<td>( \text{arcsin}(...), \text{arccos}(...), \ldots )</td>
</tr>
<tr>
<td>( \Delta List() )</td>
<td>( \text{deltaList}(...) )</td>
</tr>
</tbody>
</table>

**From the Computer Keyboard**

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i ) (imaginary constant)</td>
<td>( @i )</td>
</tr>
<tr>
<td>( e ) (natural log base e)</td>
<td>( @e )</td>
</tr>
<tr>
<td>( \varepsilon ) (scientific notation)</td>
<td>( @\varepsilon )</td>
</tr>
<tr>
<td>( T ) (transpose)</td>
<td>( @t )</td>
</tr>
<tr>
<td>( \Gamma ) (radians)</td>
<td>( @\Gamma )</td>
</tr>
<tr>
<td>( \circ ) (degrees)</td>
<td>( @\circ )</td>
</tr>
<tr>
<td>To enter this:</td>
<td>Type this shortcut:</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>g (gradians)</td>
<td>@g</td>
</tr>
<tr>
<td>∡ (angle)</td>
<td>@&lt;</td>
</tr>
<tr>
<td>➤ (conversion)</td>
<td>@&gt;</td>
</tr>
<tr>
<td>Decimal, ➤approxFraction(), and so on.</td>
<td>➤Decimal, ➤approxFraction(), and so on.</td>
</tr>
</tbody>
</table>
EOS™ (Equation Operating System) Hierarchy

This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire™ math and science learning technology. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS™ software evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

Order of Evaluation

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses ( ), brackets [ ], braces { }</td>
</tr>
<tr>
<td>2</td>
<td>Indirection (#)</td>
</tr>
<tr>
<td>3</td>
<td>Function calls</td>
</tr>
<tr>
<td>4</td>
<td>Post operators: degrees-minutes-seconds (°’,”,’), factorial (!), percentage (%), radian ( ′ ), subscript ([ ]), transpose (ᵀ)</td>
</tr>
<tr>
<td>5</td>
<td>Exponentiation, power operator (^)</td>
</tr>
<tr>
<td>6</td>
<td>Negation (¯)</td>
</tr>
<tr>
<td>7</td>
<td>String concatenation (&amp;)</td>
</tr>
<tr>
<td>8</td>
<td>Multiplication (×), division (/)</td>
</tr>
<tr>
<td>9</td>
<td>Addition (+), subtraction (−)</td>
</tr>
<tr>
<td>10</td>
<td>Equality relations: equal (=), not equal (≠ or /=), less than (&lt;), less than or equal (≤ or &lt;=), greater than (&gt;), greater than or equal (≥ or &gt;=)</td>
</tr>
<tr>
<td>11</td>
<td>Logical not</td>
</tr>
<tr>
<td>12</td>
<td>Logical and</td>
</tr>
<tr>
<td>13</td>
<td>Logical or</td>
</tr>
<tr>
<td>14</td>
<td>xor, nor, nand</td>
</tr>
<tr>
<td>15</td>
<td>Logical implication (⇒)</td>
</tr>
<tr>
<td>16</td>
<td>Logical double implication, XNOR (⇔)</td>
</tr>
<tr>
<td>17</td>
<td>Constraint operator (“</td>
</tr>
<tr>
<td>18</td>
<td>Store (→)</td>
</tr>
</tbody>
</table>

Parentheses, Brackets, and Braces

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression 4(1+2), EOS™ software first evaluates the portion of the expression inside the parentheses, 1+2, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, (1+2)/(3+4 will display the error message “Missing”).
Note: Because the TI-Nspire™ software allows you to define your own functions, a variable name followed by an expression in parentheses is considered a “function call” instead of implied multiplication. For example, \(a(b+c)\) is the function \(a\) evaluated by \(b+c\). To multiply the expression \(b+c\) by the variable \(a\), use explicit multiplication: \(a \cdot (b+c)\).

### Indirection

The indirection operator (#) converts a string to a variable or function name. For example, #("x" & "y" & "z") creates the variable name xyz. Indirection also allows the creation and modification of variables from inside a program. For example, if 10→r and "r"→s1, then #s1=10.

### Post Operators

Post operators are operators that come directly after an argument, such as 5!, 25%, or 60°15'45". Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression \(4^3!\), 3! is evaluated first. The result, 6, then becomes the exponent of 4 to yield 4096.

### Exponentiation

Exponentiation (^) and element-by-element exponentiation (.^) are evaluated from right to left. For example, the expression \(2^3^2\) is evaluated the same as \(2^{(3^2)}\) to produce 512. This is different from \((2^3)^2\), which is 64.

### Negation

To enter a negative number, press \(\text{[(-)}\) followed by the number. Post operations and exponentiation are performed before negation. For example, the result of \(-x^2\) is a negative number, and \(-9^2 = -81\). Use parentheses to square a negative number such as \((-9)^2\) to produce 81.

### Constraint ("|"")

The argument following the constraint ("|"") operator provides a set of constraints that affect the evaluation of the argument preceding the operator.
**Error Codes and Messages**

When an error occurs, its code is assigned to variable `errCode`. User-defined programs and functions can examine `errCode` to determine the cause of an error. For an example of using `errCode`, see Example 2 under the `Try` command, page 106.

**Note:** Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A function did not return a value</td>
</tr>
<tr>
<td>20</td>
<td>A test did not resolve to TRUE or FALSE. Generally, undefined variables cannot be compared. For example, the test if a&lt;b will cause this error if either a or b is undefined when the if statement is executed.</td>
</tr>
<tr>
<td>30</td>
<td>Argument cannot be a folder name.</td>
</tr>
<tr>
<td>40</td>
<td>Argument error</td>
</tr>
<tr>
<td>50</td>
<td>Argument mismatch Two or more arguments must be of the same type.</td>
</tr>
<tr>
<td>60</td>
<td>Argument must be a Boolean expression or integer</td>
</tr>
<tr>
<td>70</td>
<td>Argument must be a decimal number</td>
</tr>
<tr>
<td>90</td>
<td>Argument must be a list</td>
</tr>
<tr>
<td>100</td>
<td>Argument must be a matrix</td>
</tr>
<tr>
<td>130</td>
<td>Argument must be a string</td>
</tr>
<tr>
<td>140</td>
<td>Argument must be a variable name. Make sure that the name: • does not begin with a digit • does not contain spaces or special characters • does not use underscore or period in invalid manner • does not exceed the length limitations See the Calculator section in the documentation for more details.</td>
</tr>
<tr>
<td>160</td>
<td>Argument must be an expression</td>
</tr>
<tr>
<td>165</td>
<td>Batteries too low for sending or receiving Install new batteries before sending or receiving.</td>
</tr>
<tr>
<td>170</td>
<td>Bound The lower bound must be less than the upper bound to define the search interval.</td>
</tr>
<tr>
<td>180</td>
<td>Break The <code>esc</code> or <code>on</code> key was pressed during a long calculation or during program execution.</td>
</tr>
<tr>
<td>190</td>
<td>Circular definition This message is displayed to avoid running out of memory during infinite replacement of variable values during simplification. For example, a+1-&gt;a, where a is an undefined variable, will cause this error.</td>
</tr>
<tr>
<td>200</td>
<td>Constraint expression invalid For example, `solve(3x^2-4=0,x)</td>
</tr>
<tr>
<td>210</td>
<td>Invalid Data type An argument is of the wrong data type.</td>
</tr>
<tr>
<td>220</td>
<td>Dependent limit</td>
</tr>
</tbody>
</table>

Note: Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>Dimension Error. Not enough elements in the lists.</td>
</tr>
<tr>
<td>235</td>
<td>Dimension Error. Not enough elements in the lists.</td>
</tr>
<tr>
<td>240</td>
<td>Dimension mismatch Two or more arguments must be of the same dimension. For example, ([1,2]+[1,2,3]) is a dimension mismatch because the matrices contain a different number of elements.</td>
</tr>
<tr>
<td>250</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>260</td>
<td>Domain error An argument must be in a specified domain. For example, (\text{rand}(0)) is not valid.</td>
</tr>
<tr>
<td>270</td>
<td>Duplicate variable name</td>
</tr>
<tr>
<td>280</td>
<td>Else and ElseIf invalid outside of If..EndIf block</td>
</tr>
<tr>
<td>290</td>
<td>EndTry is missing the matching Else statement</td>
</tr>
<tr>
<td>295</td>
<td>Excessive iteration</td>
</tr>
<tr>
<td>300</td>
<td>Expected 2 or 3-element list or matrix</td>
</tr>
<tr>
<td>310</td>
<td>The first argument of (\text{nSolve}) must be an equation in a single variable. It cannot contain a non-valued variable other than the variable of interest.</td>
</tr>
<tr>
<td>320</td>
<td>First argument of solve or cSolve must be an equation or inequality For example, solve((3\times^2-4, x)) is invalid because the first argument is not an equation.</td>
</tr>
<tr>
<td>345</td>
<td>Inconsistent units</td>
</tr>
<tr>
<td>350</td>
<td>Index out of range</td>
</tr>
<tr>
<td>360</td>
<td>Indirection string is not a valid variable name</td>
</tr>
<tr>
<td>380</td>
<td>Undefined Ans Either the previous calculation did not create Ans, or no previous calculation was entered.</td>
</tr>
<tr>
<td>390</td>
<td>Invalid assignment</td>
</tr>
<tr>
<td>400</td>
<td>Invalid assignment value</td>
</tr>
<tr>
<td>410</td>
<td>Invalid command</td>
</tr>
<tr>
<td>430</td>
<td>Invalid for the current mode settings</td>
</tr>
<tr>
<td>435</td>
<td>Invalid guess</td>
</tr>
<tr>
<td>440</td>
<td>Invalid implied multiply For example, (x(x+1)) is invalid; whereas, (x^* (x+1)) is the correct syntax. This is to avoid confusion between implied multiplication and function calls.</td>
</tr>
<tr>
<td>450</td>
<td>Invalid in a function or current expression Only certain commands are valid in a user-defined function.</td>
</tr>
<tr>
<td>490</td>
<td>Invalid in Try..EndTry block</td>
</tr>
<tr>
<td>510</td>
<td>Invalid list or matrix</td>
</tr>
<tr>
<td>550</td>
<td>Invalid outside function or program A number of commands are not valid outside a function or program. For example, (\text{Local}) cannot be used unless it is in a function or program.</td>
</tr>
<tr>
<td>560</td>
<td>Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks For example, the Exit command is valid only inside these loop blocks.</td>
</tr>
<tr>
<td>565</td>
<td>Invalid outside program</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 570        | Invalid pathname  
For example, \( i \)var is invalid. |
| 575        | Invalid polar complex |
| 580        | Invalid program reference  
Programs cannot be referenced within functions or expressions such as \( 1+p(x) \) where \( p \) is a program. |
| 600        | Invalid table |
| 605        | Invalid use of units |
| 610        | Invalid variable name in a Local statement |
| 620        | Invalid variable or function name |
| 630        | Invalid variable reference |
| 640        | Invalid vector syntax |
| 650        | Link transmission  
A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends. |
| 665        | Matrix not diagonalizable |
| 670        | Low Memory  
1. Delete some data in this document  
2. Save and close this document  
If 1 and 2 fail, pull out and re-insert batteries |
| 672        | Resource exhaustion |
| 673        | Resource exhaustion |
| 680        | Missing ( |
| 690        | Missing ) |
| 700        | Missing “ |
| 710        | Missing ] |
| 720        | Missing ) |
| 730        | Missing start or end of block syntax |
| 740        | Missing Then in the If..Endif block |
| 750        | Name is not a function or program |
| 765        | No functions selected |
| 780        | No solution found |
| 800        | Non-real result  
For example, if the software is in the Real setting, \( \sqrt{-1} \) is invalid.  
To allow complex results, change the "Real or Complex" Mode Setting to RECTANGULAR or POLAR. |
| 830        | Overflow |
| 850        | Program not found  
A program reference inside another program could not be found in the provided path during execution. |
<p>| 855        | Rand type functions not allowed in graphing |
| 860        | Recursion too deep |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>870</td>
<td>Reserved name or system variable</td>
</tr>
<tr>
<td>900</td>
<td>Argument error&lt;br&gt;Median-median model could not be applied to data set.</td>
</tr>
<tr>
<td>910</td>
<td>Syntax error</td>
</tr>
<tr>
<td>920</td>
<td>Text not found</td>
</tr>
<tr>
<td>930</td>
<td>Too few arguments&lt;br&gt;The function or command is missing one or more arguments.</td>
</tr>
<tr>
<td>940</td>
<td>Too many arguments&lt;br&gt;The expression or equation contains an excessive number of arguments and cannot be evaluated.</td>
</tr>
<tr>
<td>950</td>
<td>Too many subscripts</td>
</tr>
<tr>
<td>955</td>
<td>Too many undefined variables</td>
</tr>
<tr>
<td>960</td>
<td>Variable is not defined&lt;br&gt;No value is assigned to variable. Use one of the following commands:&lt;br&gt;• sto ➔&lt;br&gt;• :=&lt;br&gt;• Define&lt;br&gt;to assign values to variables.</td>
</tr>
<tr>
<td>965</td>
<td>Unlicensed OS</td>
</tr>
<tr>
<td>970</td>
<td>Variable in use so references or changes are not allowed</td>
</tr>
<tr>
<td>980</td>
<td>Variable is protected</td>
</tr>
<tr>
<td>990</td>
<td>Invalid variable name&lt;br&gt;Make sure that the name does not exceed the length limitations</td>
</tr>
<tr>
<td>1000</td>
<td>Window variables domain</td>
</tr>
<tr>
<td>1010</td>
<td>Zoom</td>
</tr>
<tr>
<td>1020</td>
<td>Internal error</td>
</tr>
<tr>
<td>1030</td>
<td>Protected memory violation</td>
</tr>
<tr>
<td>1040</td>
<td>Unsupported function. This function requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1045</td>
<td>Unsupported operator. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1050</td>
<td>Unsupported feature. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1060</td>
<td>Input argument must be numeric. Only inputs containing numeric values are allowed.</td>
</tr>
<tr>
<td>1070</td>
<td>Trig function argument too big for accurate reduction</td>
</tr>
<tr>
<td>1080</td>
<td>Unsupported use of Ans. This application does not support Ans.</td>
</tr>
<tr>
<td>1090</td>
<td>Function is not defined. Use one of the following commands:&lt;br&gt;• Define&lt;br&gt;• :=&lt;br&gt;• sto ➔&lt;br&gt;to define a function.</td>
</tr>
<tr>
<td>1100</td>
<td>Non-real calculation&lt;br&gt;For example, if the software is in the Real setting, (\sqrt{-1}) is invalid.&lt;br&gt;To allow complex results, change the &quot;Real or Complex&quot; Mode Setting to RECTANGULAR or POLAR.</td>
</tr>
<tr>
<td>1110</td>
<td>Invalid bounds</td>
</tr>
<tr>
<td>1120</td>
<td>No sign change</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1130</td>
<td>Argument cannot be a list or matrix</td>
</tr>
<tr>
<td>1140</td>
<td>Argument error&lt;br&gt;The first argument must be a polynomial expression in the second argument. If the second argument is omitted, the software attempts to select a default.</td>
</tr>
<tr>
<td>1150</td>
<td>Argument error&lt;br&gt;The first two arguments must be polynomial expressions in the third argument. If the third argument is omitted, the software attempts to select a default.</td>
</tr>
<tr>
<td>1160</td>
<td>Invalid library pathname&lt;br&gt;A pathname must be in the form <code>xxx\yyy</code>, where:&lt;br&gt;- The <code>xxx</code> part can have 1 to 16 characters.&lt;br&gt;- The <code>yyy</code> part can have 1 to 15 characters.&lt;br&gt;See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1170</td>
<td>Invalid use of library pathname&lt;br&gt;- A value cannot be assigned to a pathname using Define, :=, or sto →.&lt;br&gt;- A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition.</td>
</tr>
<tr>
<td>1180</td>
<td>Invalid library variable name.&lt;br&gt;Make sure that the name:&lt;br&gt;- Does not contain a period&lt;br&gt;- Does not begin with an underscore&lt;br&gt;- Does not exceed 15 characters&lt;br&gt;See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1190</td>
<td>Library document not found:&lt;br&gt;- Verify library is in the MyLib folder.&lt;br&gt;- Refresh Libraries.&lt;br&gt;See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1200</td>
<td>Library variable not found:&lt;br&gt;- Verify library variable exists in the first problem in the library.&lt;br&gt;- Make sure library variable has been defined as LibPub or LibPriv.&lt;br&gt;- Refresh Libraries.&lt;br&gt;See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1210</td>
<td>Invalid library shortcut name.&lt;br&gt;Make sure that the name:&lt;br&gt;- Does not contain a period&lt;br&gt;- Does not begin with an underscore&lt;br&gt;- Does not exceed 16 characters&lt;br&gt;- Is not a reserved name&lt;br&gt;See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1220</td>
<td>Domain error:&lt;br&gt;The tangentLine and normalLine functions support real-valued functions only.</td>
</tr>
<tr>
<td>1230</td>
<td>Domain error:&lt;br&gt;Trigonometric conversion operators are not supported in Degree or Gradian angle modes.</td>
</tr>
</tbody>
</table>
| 1250       | Argument Error<br>Use a system of linear equations.<br>Example of a system of two linear equations with variables x and y:<br>\[ 3x + 7y = 5 \]
\[ 2y - 5x = -1 \] |
<p>| 1260       | Argument Error&lt;br&gt;The first argument of <code>nfMin</code> or <code>nfMax</code> must be an expression in a single variable. It cannot contain a non-valued variable other than the variable of interest. |
| 1270       | Argument Error&lt;br&gt;Order of the derivative must be equal to 1 or 2. |
| 1280       | Argument Error&lt;br&gt;Use a polynomial in expanded form in one variable. |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1290       | Argument Error  
Use a polynomial in one variable. |
| 1300       | Argument Error  
The coefficients of the polynomial must evaluate to numeric values. |
| 1310       | Argument error:  
A function could not be evaluated for one or more of its arguments. |
| 1380       | Argument error:  
Nested calls to domain() function are not allowed. |
Warning Codes and Messages

You can use the `warnCodes()` function to store the codes of warnings generated by evaluating an expression. This table lists each numeric warning code and its associated message. For an example of storing warning codes, see `warnCodes()`, page 111.

<table>
<thead>
<tr>
<th>Warning code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Operation might introduce false solutions.</td>
</tr>
<tr>
<td>10001</td>
<td>Differentiating an equation may produce a false equation.</td>
</tr>
<tr>
<td>10002</td>
<td>Questionable solution</td>
</tr>
<tr>
<td>10003</td>
<td>Questionable accuracy</td>
</tr>
<tr>
<td>10004</td>
<td>Operation might lose solutions.</td>
</tr>
<tr>
<td>10005</td>
<td>cSolve might specify more zeros.</td>
</tr>
<tr>
<td>10006</td>
<td>Solve may specify more zeros.</td>
</tr>
</tbody>
</table>
| 10007        | More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess. Examples using `solve()`:
  • `solve(Equation, Var=Guess)`, `lowBound<Var<upBound`
  • `solve(Equation, Var)`, `lowBound<Var<upBound`
  • `solve(Equation, Var=Guess)` |
| 10008        | Domain of the result might be smaller than the domain of the input. |
| 10009        | Domain of the result might be larger than the domain of the input. |
| 10012        | Non-real calculation |
| 10013        | `∞^0` or `undef^0` replaced by 1 |
| 10014        | `undef^0` replaced by 1 |
| 10015        | `1^∞` or `1^undef` replaced by 1 |
| 10016        | `1^undef` replaced by 1 |
| 10017        | Overflow replaced by `∞` or `~∞` |
| 10018        | Operation requires and returns 64 bit value. |
| 10019        | Resource requires and returns 64 bit value. |
| 10020        | Trig function argument too big for accurate reduction. |
| 10021        | Input contains an undefined parameter. Result might not be valid for all possible parameter values. |
| 10022        | Specifying appropriate lower and upper bounds might produce a solution. |
| 10023        | Scalar has been multiplied by the identity matrix. |
| 10024        | Result obtained using approximate arithmetic. |
| 10025        | Equivalence cannot be verified in EXACT mode. |
| 10026        | Constraint might be ignored. Specify constraint in the form "\"’Value MathTestSymbol Constant’ or a conjunct of these forms, for example ‘x<3 and x>=12’"
Service and Support

Texas Instruments Support and Service

For U.S. and Canada:

For General Information

Home Page: education.ti.com
KnowledgeBase and e-mail inquiries: education.ti.com/support
Phone: (800) TI-CARES / (800) 842-2737
For U.S., Canada, Mexico, Puerto Rico, and Virgin Islands only
International information: education.ti.com/international

For Technical Support

KnowledgeBase and support by e-mail: education.ti.com/support
Phone (not toll-free): (972) 917-8324

For Product (Hardware) Service

Customers in the U.S., Canada, Mexico, Puerto Rico and Virgin Islands: Always contact Texas Instruments Customer Support before returning a product for service.

For All Other Countries:

For general information

For more information about TI products and services, contact TI by e-mail or visit the TI Internet address.

E-mail inquiries: ti-cares@ti.com
Home Page: education.ti.com
**Service and Warranty Information**

For information about the length and terms of the warranty or about product service, refer to the warranty statement enclosed with this product or contact your local Texas Instruments retailer/distributor.
Index

Symbols

|^, power 119
|^−1, reciprocal 129
:=, assign 131
!, factorial 124
.^, dot power 120
.*., dot multiplication 120
.+., dot addition 120
.-., dot subtraction 120
.-., dot division 120
', minute notation 128
", second notation 128
≤, less than or equal 123
©, comment 131
Δlist(), list difference 55
°, degree notation 128
°, degrees/minutes/seconds 128
∫, integral 125
√, square root 125
≠, not equal 122
−, subtract 117
÷, divide 118
Π, product 125
Σ( ), sum 126
⇔, logical double implication 124
⇒, logical implication 123, 134
*, multiply 118
&, append 124
→, store 130
#, indirection 127
#, indirection operator 137
%, percent 121
+, add 117
<, less than 122
=, equal 122
>, greater than 123
|, constraint operator 130
≥, greater than or equal 123

Numerics

0b, binary indicator 131
0h, hexadecimal indicator 131
10^( ), power of ten 129
2-sample F Test 39

▸approxFraction( ) 10

A

abs( ), absolute value 6
absolute value
  template for 3
add, + 117
amortization table, amortTbl( ) 6, 12
amortTbl( ), amortization table 6, 12
and, Boolean operator 6
angle, angle( ) 7
angle( ), angle 7
ANOVA, one-way variance analysis 7
ANOVA2way, two-way variance analysis 8
Ans, last answer 9
answer (last), Ans 9
append, & 124
approx( ), approximate 10
approximate, approx( ) 10
approxRational( ) 10
arccos() 10
arccosh() 10
arccot() 10
arccoth() 11
arccsc() 11
arccsch() 11
arcsec() 11
arcsech() 11
arcsin() 10
arcsinh() 11
arctan() 11
arctan() 11
arctan( ), arctan( ) 102
arctanh() 11
arguments in TVM functions 108
augment( ), augment/concatenate 11
augment/concatenate, augment( ) 11
average rate of change, avgRC( ) 12
avgRC( ), average rate of change 12
B
Base10, display as decimal integer 13
Base16, display as hexadecimal 14
Base2, display as binary 12
binary
   display, ▶Base2 12
   indicator, 0b 131
binomCdf( ) 14
binomPdf( ) 14
Boolean operators
   and 6
   nand 66
   nor 69
   not 70
   or 73
   ⇔ 124
   xor 112
   ⇒ 123, 134

C
χ²2way 15
χ²Cdf( ) 16
χ²GOF 16
χ²Pdf( ) 16
Cdf( ) 36
celling, ceiling( ) 14, 15, 23
celling( ), ceiling 14
centralDiff( ) 15
char( ), character string 15
character string, char( ) 15
characters
   numeric code, ord( ) 73
   string, char( ) 15
clear
   error, ClrErr 17
ClearAZ 16
ClrErr, clear error 17
colAugment 17
colDim( ), matrix column dimension 17
colNorm( ), matrix column norm 17
combinations, nCr( ) 67
comment, © 131
completeSquare( ), complete square 18
complex
   conjugate, conj( ) 18
   conj(), complex conjugate 18
   constraint operator "|" 130
   constraint operator, order of evaluation 136
   construct matrix, constructMat( ) 18
   constructMat( ), construct matrix 18
   convert
   ▶Grad 44
   ▶Rad 81
copy variable or function, CopyVar 18
correlation matrix, corrMat( ) 19
corrMat( ), correlation matrix 19
cos( ), cosine 19
cos⁻¹, arccosine 20
cosh( ), hyperbolic cosine 21
cosh⁻¹(), hyperbolic arccosine 21
cosine, cos( ) 19
cot( ), cotangent 21
cot⁻¹( ), arccotangent 22
cotangent, cot( ) 21
coth( ), hyperbolic cotangent 22
coth⁻¹( ), hyperbolic arccotangent 22
count days between dates, dbd( ) 26
count items in a list conditionally ,
countif( ) 23
count items in a list, count( ) 22
count( ), count items in a list 22
countif(), conditionally count items
   in a list 23
cPolyRoots( ) 23
cross product, crossP( ) 23
crossP( ), cross product 23
csc( ), cosecant 24
csc⁻¹( ), inverse cosecant 24
csch( ), hyperbolic cosecant 24
csch⁻¹( ), inverse hyperbolic cosecant 24
cubic regression, CubicReg 25
CubicReg, cubic regression 25
cumulative sum, cumulativeSum( ) 25
cumulativeSum( ), cumulative sum 25
cycle, Cycle 26
cycle, Cycle 26
Cylind, display as cylindrical vector 26.
cylindrical vector display, Cylind 26.

D

d ( ), first derivative 124
days between dates, dbd ( ) 26
dbd ( ), days between dates 26
DD, display as decimal angle 27
Decimal, display result as decimal 27
decimal
angle display, DD 27
integer display, Base10 13
Define 27
Define LibPriv 28
Define LibPub 28
Define, define 27
define, Define 27
defining
private function or program 28
public function or program 28
definite integral
template for 5
degree notation, ° 128
degree/minute/second display, DMS 31
degree/minute/second notation 128
delete
void elements from list 29
deleting
variable, DelVar 29
deltaList ( ) 29
DelVar, delete variable 29
delVoid ( ), remove void elements 29
derivatives
first derivative, d ( ) 124
numeric derivative, nDeriv ( ) 68
numeric derivative,
nDerivative ( ) 67
det ( ), matrix determinant 29
diag ( ), matrix diagonal 30
dim ( ), dimension 30
dimension, dim ( ) 30
Disp, display data 30
display as
binary, Base2 12
cylindrical vector, Cylind 26
decimal angle, DD 27
decimal integer, Base10 13
degree/minute/second, DMS 31
hexadecimal, Base16 14
polar vector, Polar 75
rectangular vector, Rect 82
spherical vector, Sphere 97
display data, Disp 30
distribution functions
binomCdf ( ) 14
binomPdf ( ) 14
χ² 2way ( ) 15
χ² Cdf ( ) 16
χ² GOF ( ) 16
χ² Pdf ( ) 16
Invχ² ( ) 48
invNorm ( ) 48
invt ( ) 48
normCdf ( ) 69
normPdf ( ) 69
poissCdf ( ) 74
poissPdf ( ) 74
tCdf ( ) 104
tPdf ( ) 105
divide, ÷ 118
DMS, display as degree/minute/
second 31
dot
addition, .+ 120
division, .÷ 120
multiplication, .* 120
power, .^ 120
product, dotP ( ) 31
subtraction, .− 120
dotP ( ), dot product 31

E
e exponent
template for 2
e to a power, e^ ( ) 31, 35
e^ ( ), e to a power 31
E, exponent 127
eff ( ), convert nominal to effective
rate 32
effective rate, eff ( ) 32
eigenvalue, eigVl ( ) 32
eigenvector, eigVc( ) 32
eigVc( ), eigenvector 32
eigVl( ), eigenvalue 32
else if, ElseIf 33
else, Else 45
ElseIf, else if 33
empty (void) elements 132
end
for, EndFor 38
function, EndFunc 40
if, EndIf 45
loop, EndLoop 60
program, EndPrgm 77
try, EndTry 106
while, EndWhile 112
end function, EndFunc 40
end if, EndIf 45
end loop, EndLoop 60
end while, EndWhile 112
EndTry, end try 106
EndWhile, end while 112
EOS (Equation Operating System) 136
equal, = 122
Equation Operating System (EOS) 136
error codes and messages 138
errors and troubleshooting
    clear error, ClrErr 17
    pass error, PassErr 74
euler( ), Euler function 34
evaluate polynomial, polyEval( ) 75
evaluation, order of 136
exclusion with "|" operator 130
Exit, exit 34
exit, Exit 34
exp( ), e to a power 35
exponent, E 127
exponential regression, ExpReg 35
exponents
    template for 1
expr( ), string to expression 35
ExpReg, exponential regression 35
expressions
    string to expression, expr( ) 35

F
factor, factor() 36
factor(), factor 36
factorial, ! 124
Fill, matrix fill 36
financial functions, tvmFV( ) 107
financial functions, tvmI( ) 108
financial functions, tvmN( ) 108
financial functions, tvmPmt( ) 108
financial functions, tvmPV( ) 108
first derivative
    template for 5
FiveNumSummary 37
floor, floor() 37
floor(), floor 37
For 38
For, for 38
for, For 38
format string, format() 38
format(), format string 38
fpart( ), function part 38
fractions
    propFrac 78
    template for 1
freqTable() 39
frequency( ) 39
Frobenius norm, norm( ) 69
Func, function 40
Func, program function 40
functions
    part, fpart( ) 38
    program function, Func 40
    user-defined 27
functions and variables
    copying 18

G
g, gradians 128
gcd(), greatest common divisor 40
gemCdf( ) 41
geomPdf( ) 41
get/return
    denominator, getDenom( ) 41
    number, getNum( ) 43
variables information,
    getVarInfo( ) 41, 43
getDenom( ), get/return
denominator 41
getLangInfo( ), get/return language
information 41
getLockInfo( ), tests lock status of
variable or variable group 42
getMode( ), get mode settings 42
getNum( ), get/return number 43
getType( ), get type of variable 43
getVarInfo( ), get/return variables
information 43
go to, Goto 44
Goto, go to 44
•, convert to gradian angle 44
gradian notation, \( g \) 128
greater than or equal, \( \geq \) 123
greater than, \( > \) 123
greatest common divisor, \( \gcd() \) 40
groups, locking and unlocking 57, 110
groups, testing lock status 42

H

hexadecimal
display, \( \text{Base16} \) 14
indicator, \( 0h \) 131
hyperbolic
arccosine, \( \cosh^{-1}() \) 21
arcsine, \( \sinh^{-1}() \) 95
arctangent, \( \tanh^{-1}() \) 103
cosine, \( \cosh() \) 21
sine, \( \sinh() \) 95
tangent, \( \tanh() \) 103

I

identity matrix, identity( ) 45
identity( ), identity matrix 45
If, if 45
if, If 45
ifFn( ) 46
imag( ), imaginary part 46
imaginary part, imag( ) 46
indirection operator (#) 137
indirection, \( # \) 127
inString( ), within string 47
int( ), integer 47
intDiv( ), integer divide 47
integer divide, intDiv( ) 47
integer part, iPart( ) 49
integer, int( ) 47
integral, \( \int \) 125
interpolate( ), interpolate 48
\( \text{Inv}\chi^2() \) 48
inverse cumulative normal
distribution (invNorm( ) 48
inverse, \( ^{-1} \) 129
invF( ) 48
invNorm( ), inverse cumulative
normal distribution) 48
invt( ) 48
iPart( ), integer part 49
irr( ), internal rate of return
internal rate of return, irr( ) 49
isPrime( ), prime test 49
isVoid( ), test for void 49

L

label, Lbl 50
language
get language information 41
Lbl, label 50
lcm, least common multiple 50
least common multiple, \( \text{lcm} \) 50
left, left( ) 50
left( ), left 50
length of string 30
less than or equal, \( \leq \) 123
less than, \( < \) 122
LibPriv 28
LibPub 28
library
create shortcuts to objects 51
libShortcut( ), create shortcuts to
library objects 51
linear regression, LinRegAx 52
linear regression, LinRegBx 51, 52
LinRegBx, linear regression 51
LinRegMx, linear regression 52
LinRegtIntervals, linear regression 52
LinRegtTest 54
linSolve() 55
list to matrix, list\( \text{mat}() \) 55
list, conditionally count items in 23
list, count items in 22
list→mat( ), list to matrix 55
lists
  augment/concatenate,
  augment() 11
cross product, crossP( ) 23
cumulative sum,
  cumulativeSum( ) 25
difference, Δlist( ) 55
differences in a list, Δlist( ) 55
dot product, dotP( ) 31
empty elements in 132
list to matrix, list→mat( ) 55
matrix to list, mat→list( ) 60
maximum, max() 61
mid-string, mid( ) 62
minimum, min() 63
new, newList( ) 67
product, product( ) 77
sort ascending, SortA 96
sort descending, SortD 97
summation, sum() 100, 101
ln( ), natural logarithm 55
LnReg, logarithmic regression 56
local variable, Local 57
local, Local 57
Local, local variable 57
Lock, lock variable or variable group 57
locking variables and variable groups 57
Log
  template for 2
logarithmic regression, LnReg 56
logarithms 55
logical double implication, ⇔ 124
logical implication, ⇒ 123, 134
logistic regression, Logistic 58
logistic regression, LogisticD 59
Logistic, logistic regression 58
LogisticD, logistic regression 59
Loop, loop 60
loop, Loop 60
LU, matrix lower-upper decomposition 60

M
mat→list( ), matrix to list 60
matrices
  augment/concatenate,
  augment() 11
column dimension, colDim( ) 17
column norm, colNorm( ) 17
cumulative sum,
  cumulativeSum( ) 25
determinant, det( ) 29
diagonal, diag( ) 30
dimension, dim( ) 30
dot addition, .+ 120
dot division, ÷ 120
dot multiplication, .* 120
dot power, .^ 120
dot subtraction, .− 120
eigenvalue, eigVl( ) 32
eigenvector, eigVc( ) 32
filling, Fill 36
identity, identity( ) 45
list to matrix, list→mat( ) 55
lower-upper decomposition, LU 60
matrix to list, mat→list( ) 60
maximum, max( ) 61
minimum, min( ) 63
new, newMat( ) 68
product, product( ) 77
QR factorization, QR 78
random, randMat( ) 81
reduced row echelon form, 
  rref( ) 88
row addition, rowAdd( ) 87
row dimension, rowDim( ) 88
row echelon form, ref( ) 83
row multiplication and addition,
  mRowAdd( ) 64
row norm, rowNorm( ) 88
row operation, mRow( ) 64
row swap, rowSwap( ) 88
submatrix, subMat( ) 100, 101
summation, sum( ) 100, 101
transpose, ↑ 101
matrix (1 × 2)
  template for 4
matrix (2 × 1)
template for 4 matrix (2 x 2)
template for 3 matrix (m x n)
template for 4 matrix to list, mat2list() 60
max(), maximum 61
maximum, max() 61
mean, mean() 61
mean(), mean 61
median, median() 61
median(), median 61
medium-medium line regression, MedMed 62
MedMed, medium-medium line regression 62
mid(), mid-string 62
mid-string, mid() 62
min(), minimum 63
minimum, min() 63
minute notation, ’ 128
mirr(), modified internal rate of return 63
mixed fractions, using propFrac() with 78
mod(), modulo 64
mode settings, getMode() 42
modes
setting, setMode() 91
modified internal rate of return, mirr() 63
modulo, mod() 64
mRow(), matrix row operation 64
mRowAdd(), matrix row multiplication and addition 64
Multiple linear regression t test 65
multiply, * 118
MultReg 64
MultRegIntervals() 65
MultRegTests() 65

N
nand, Boolean operator 66
natural logarithm, ln() 55
nCr(), combinations 67
nDerivative(), numeric derivative 67
negation, entering negative numbers 137
net present value, npv() 71
new
list, newList() 67
matrix, newMat() 68
newList(), new list 67
newMat(), new matrix 68
nfMax(), numeric function maximum 68
nfMin(), numeric function minimum 68
nInt(), numeric integral 68
nom(), convert effective to nominal rate 69
nominal rate, nom() 69
nor, Boolean operator 69
norm(), Frobenius norm 69
normal distribution probability,
normCdf() 69
normCdf() 69
normPdf() 69
not equal, 122
not, Boolean operator 70
nPr(), permutations 70
npv(), net present value 71
nSolve(), numeric solution 71
nth root
setting, setMode() 91
numeric
derivative, nDeriv() 68
derivative, nDerivative() 67
integral, nInt() 68
solution, nSolve() 71
O
objects
create shortcuts to library 51
OneVar, one-variable statistics 72
one-variable statistics, OneVar 72
operators
order of evaluation 136
or (Boolean), or 73
or, Boolean operator 73
ord(), numeric character code 73
P
PRx( ), rectangular x coordinate 73
PRy( ), rectangular y coordinate 74
pass error, PassErr 74
PassErr, pass error 74
Pdf( ) 38
percent, % 121
permutations, nPr( ) 70
piecewise function (2-piece) template for 2
piecewise function (N-piece) template for 2
piecewise( ) 74
poissCdf( ) 74
poissPdf( ) 74
Polar, display as polar vector 75
polar coordinate, Rθ( ) 80
coordinate, Rr( ) 80
vector display, Polar 75
polyEval( ), evaluate polynomial 75
polynomials evaluate, polyEval( ) 75
random, randPoly( ) 82
PolyRoots() 75
power of ten, 10^( ) 129
power regression, PowerReg 75, 76, 84, 85, 104
power, ^ 119
PowerReg, power regression 76
Prgm, define program 77
prime number test, isPrime( ) 49
probability density, normPdf( ) 69
prodSeq() 77
product (Π) template for 4
product, Π( ) 125
product, product( ) 77
product( ), product 77
programming define program, Prgm 77
display data, Disp 30
pass error, PassErr 74
programs
defining private library 28
defining public library 28
programs and programming
clear error, ClrErr 17
display I/O screen, Disp 30
end program, EndPrgm 77
end try, EndTry 106
try, Try 106
proper fraction, propFrac 78
propFrac, proper fraction 78
Q
QR factorization, QR 78
QR, QR factorization 78
quadratic regression, QuadReg 79
QuadReg, quadratic regression 79
quartic regression, QuartReg 79
QuartReg, quartic regression 79
R
r, radian 128
Rθ( ), polar coordinate 80
Rr( ), polar coordinate 80
Rad, convert to radian angle 81
radian, r 128
rand( ), random number 81
randBin, random number 81
randInt( ), random integer 81
randMat( ), random matrix 81
randNorm( ), random norm 81
random matrix, randMat( ) 81
norm, randNorm( ) 81
number seed, RandSeed 82
polynomial, randPoly( ) 82
random sample 82
randPoly( ), random polynomial 82
randSamp( ) 82
RandSeed, random number seed 82
real, real( ) 82
real( ), real 82
reciprocal, ^ -1 129
Rect, display as rectangular vector 82
rectangular x coordinate, PRx( ) 73
rectangular y coordinate, PRy( ) 74
rectangular-vector display, Rect 82
reduced row echelon form, rref( ) 88
ref( ), row echelon form 83
regressions
cubic, CubicReg 25
exponential, ExpReg 35
linear regression, LinRegAx 52
linear regression, LinRegBx 51, 52
logarithmic, LnReg 56
Logistic 58
logistic, Logistic 59
medium-medium line, MedMed 62
MultReg 64
power regression, PowerReg 75, 76, 84, 85, 104
quadratic, QuadReg 79
quartic, QuartReg 79
sinusoidal, SinReg 96
remain(), remainder 84
remainder, remain() 84
remove
void elements from list 29
Request 84
RequestStr 85
result values, statistics 99
results, statistics 98
Return, return 85
return, Return 85
right, right() 18, 34, 48, 85, 86, 111
right(), right 85
rk23(), Runge Kutta function 86
rotate, rotate() 86
rotate(), rotate 86
round, round() 87
round(), round 87
row echelon form, ref() 83
rowAdd(), matrix row addition 87
rowDim(), matrix row dimension 88
rowNorm(), matrix row norm 88
rowSwap(), matrix row swap 88
rref(), reduced row echelon form 88

S
sec(), secant 89
sec⁻¹(), inverse secant 89
sech(), hyperbolic secant 89
sech⁻¹(), inverse hyperbolic secant 89
second derivative
template for 5
second notation, " 128
seq(), sequence 90
seqGen(), 90
seqn() 91
sequence, seq() 90, 91
set
mode, setMode() 91
setMode(), set mode 91
settings, get current 42
shift, shift() 92
shift(), shift 92
sign, sign() 93
sign(), sign 93
simult(), simultaneous equations 93
simultaneous equations, simult() 93
sin(), sine 94
sin⁻¹(), arcsine 94
sine, sin() 94
sinh(), hyperbolic sine 95
sinh⁻¹(), hyperbolic arcsine 95
SinReg, sinusoidal regression 96
ΣInt() 126
sinusoidal regression, SinReg 96
SortA, sort ascending 96
SortD, sort descending 97
sorting
ascending, SortA 96
descending, SortD 97
Sphere, display as spherical vector 97
spherical vector display, Sphere 97
ΣPrn() 127
sqrt(), square root 97
square root
template for 1
square root, √() 97, 125
standard deviation, stdDev() 99, 110
stat.results 98
stat.values 99
statistics
combinations, nCr() 67
factorial, ! 124
mean, mean() 61
median, median() 61
one-variable statistics, OneVar 72
permutations, nPr() 70
random norm, randNorm() 81
random number seed, RandSeed
standard deviation, stdDev( )
two-variable results, TwoVar
variance, variance( )
stdDevPop( ), population standard
deviation
stdDevSamp( ), sample standard
deviation
Stop command storing
symbol, →
string
dimension, dim( )
length
string( ), expression to string
append, &
character code, ord( )
character string, char( )
expression to string, string( )
format, format( )
formatting
indirection, #
left, left( )
mid-string, mid( )
right, right( )
rotate, rotate( )
shift, shift( )
string to expression, expr( )
using to create variable names
within, InString
student-t distribution probability,
tCdf( )
student-t probability density, tPdf( )
subMat( ), submatrix
submatrix, subMat( )
substitution with "|" operator
subtract, −
sum (Σ)
template for
sum of interest payments
sum of principal payments
sum, Σ( )
time value of money, Interest 108
time value of money, number of payments 108

variance, variance( ) 111
varPop( ) 110
varSamp( ), sample variance 111
vectors
cross product, crossP( ) 23
cylindrical vector display, ▶Cylind 26
dot product, dotP( ) 31
unit, unitV( ) 110
void elements 132
void elements, remove 29
void, test for 49

X
x2, square 119
XNOR 124
xor, Boolean exclusive or 112

Z
zInterval_1Prop, one-proportion z confidence interval 113
zInterval_2Prop, two-proportion z confidence interval 114
zInterval_2Samp, two-sample z confidence interval 114
zInterval, z confidence interval 113
zTest 115
zTest_1Prop, one-proportion z test 115
zTest_2Prop, two-proportion z test 116
zTest_2Samp, two-sample z test 116